Series 2600 System SourceMeter®

Reference Manual

2600S-901-01 Rev. C / January 2008

WARRANTY

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

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

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

To exercise this warranty, write or call your local Keithley Instruments representative, or contact Keithley Instruments 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 Instruments' 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: COST 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 (1-888-534-8453) • www.keithley.com

3/07

Series 2600
System SourceMeter® Instruments
Reference Manual

©2008, Keithley Instruments, Inc.
All rights reserved.
Any unauthorized reproduction, photocopy, or use the information herein, in whole or in part without the prior written approval of Keithley Instruments, Inc. is strictly prohibited.

TSP, TSP-Link, and TSP-Net are trademarks of Keithley Instruments, Inc.
All Keithley Instruments product names are trademarks or registered trademarks of Keithley Instruments, Inc.
Other brand names are trademarks or registered trademarks of their respective holders

Cleveland, Ohio, U.S.A.
Document Number: 2600S-901-01 Rev. C / January 2008

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 2600S-901-01).....May 2006
Revision B (Document Number 2600S-901-01).....September 2007
Revision C (Document Number 2600S-901-01) .....January 2008

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

KEITHLEY

Safety Precautions

The following safety precautions should be observed before using this product and any associated instrumentation. Although some instruments and accessories would normally be used with 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 user documentation for complete product specifications.

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

The types of product users are:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The symbol on an instrument shows that it can source or measure 1000V 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 on an instrument shows that the surface may be hot. Avoid personal contact to prevent burns.

The symbol indicates a connection terminal to the equipment frame.

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

Table of Contents

1 Getting Started 1-1
Introduction 1-2
Capabilities and features 1-2
Organization of manual sections 1-3
General information 1-3
Warranty information 1-3
Contact information 1-3
Safety symbols and terms 1-3
Unpacking and inspection 1-3
Options and accessories 1-4
User's and Reference manuals 1-5
Front and rear panel familiarization 1-6
Front panel summaries 1-6
Rear panel summaries 1-9
Cooling vents 1-13
Power-up 1-14
Line power connection 1-14
Power-up sequence 1-15
Beeper 1-15
Display modes 1-16
Editing controls 1-17
Source and compliance editing 1-17
Menu navigation 1-18
Menu types 1-19
Interface selection 1-20
To select the GPIB interface 1-21
To select the RS-232 interface 1-21
Error and status messages 1-21
Default settings 1-21
Front panel setups 1-21
Remote operation setups 1-22
Remote programming 1-24
Requesting readings 1-24
Requesting command settings 1-24

2 TSP Programming 2-1
Introduction 2-3
Test Script Processor (TSP) 2-3
Run-time environment 2-3
Queries 2-4
Scripts 2-4
Named scripts 2-4
Functions 2-5
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 2-7
TSP programming levels 2-8
Programming model for scripts 2-8
Installing the Test Script Builder software 2-9
System connections 2-9

GPIB 2-9
RS-232 2-10
Using Test Script Builder 2-11
Project Navigator 2-11
Script Editor 2-11
Programming Interaction 2-11
Starting Test Script Builder 2-12
Opening communications 2-13
Creating and modifying a script 2-15
Script launch configuration 2-19
Launching a script 2-22
Running a TSP file 2-23
Retrieving scripts from the Series 2600 2-23
Instrument Console 2-24
File management tasks 2-30
Using the expanded system 2-33
Sending commands and statements 2-33
Source-measure voltage and current 2-33
Read and write to Digital I/O port 2-33
Display user-defined messages 2-34
User scripts 2-34
Script examples 2-34
Creating a user script 2-36
Saving a user script 2-37
Running a user script 2-38
Modifying a user script 2-40
Script management 2-40
Factory scripts 2-42
Running a factory script 2-42
Modifying a factory script 2-42
Differences: Remote versus local state 2-43
Memory considerations for the runtime environment 2-44
Test Script Language (TSL) reference 2-45
Introduction 2-45
Reserved words 2-45
Variables and types 2-45
Operators 2-46
Functions 2-46
Tables/arrays 2-47
Precedence 2-48
Logical operators 2-48
Concatenation 2-49
Branching 2-50
Loop control 2-51
Standard libraries 2-53

3 DUT Test Connections 3-1
Input/output connectors 3-2
Input/output LO and chassis ground 3-3
Sensing methods 3-5
2-wire local sensing 3-6
4-wire remote sensing 3-7
Sense mode selection 3-8
Contact check connections 3-8
Multiple SMU connections 3-10
Guarding and shielding 3-12
Guarding 3-12
Noise shield 3-13
Safety shield 3-16
Using shielding and guarding together 3-18
Test fixture 3-19
Floating a SMU 3-20
Output-off states 3-23
Selecting the Output-off state 3-23

4 Basic Operation 4-1
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 4-6
Warm-up 4-6
Auto zero 4-6
NPLC caching 4-6
Basic source-measure procedure 4-8
Front panel source-measure procedure 4-8
Remote source-measure procedure 4-9
Measure only 4-11
Sink operation 4-12
Ohms measurements 4-12
Ohms calculations 4-12
Ohms ranging 4-12
Basic ohms measurement procedure 4-12
Ohms sensing 4-13
Sense selection 4-14
Remote ohms programming 4-15
Power measurements 4-16
Power calculations 4-16
Basic power measurement procedure 4-16
Remote power programming 4-16
Contact check measurements 4-17
Overview 4-17
Contact check commands 4-18
Contact check programming example 4-19

5 Sweep Operation 5-1
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 5-6
Sweep measurement storage 5-7
Sweep functions 5-7
Staircase sweep functions 5-8
Pulse sweep functions 5-8
Custom sweep functions 5-9
Running sweeps 5-9
Front panel 5-9
Sweep programming examples 5-9

6 Range, Digits, Speed, Rel, and Filters 6-1
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 6-3
Range considerations 6-4
Range programming 6-4
Digits 6-6
Setting display resolution 6-6
Remote digits programming 6-6

Speed 6-6
Setting speed 6-7
Remote speed programming 6-7
Rel 6-8
Front panel rel 6-8
Remote rel programming 6-9
Filters 6-10
Filter types 6-10
Response time considerations 6-10
Front panel filter control 6-10
Remote filter programming 6-13

7 Buffer (Data Store) 7-1
Overview 7-2
Data store overview 7-2
Front panel data store 7-2
Buffer configuration 7-2
Storing readings 7-3
Recalling readings 7-3
Remote data store 7-4
Data store commands 7-4
Reading buffers 7-5
Time and date values 7-7
Buffer status 7-8
Dynamically allocated buffers 7-8
Buffer programming examples 7-9

8 Source-Measure Concepts 8-1
Overview 8-2
Compliance limit 8-2
Maximum compliance 8-2
Compliance principles 8-3
Sweep waveforms 8-3
Staircase sweeps 8-3
Pulse sweeps 8-4
Overheating protection 8-4
Power equations to avoid overheating 8-4
Operating boundaries 8-7
Source or sink 8-7
Continuous power operating boundaries 8-8
I-Source operating boundaries 8-9
V-Source operating boundaries 8-13
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) 8-17
Contact check 8-18
Guard 8-19
Guard overview 8-19
Guard connections 8-20
Pulse concepts 8-22
Pulse period 8-22
Pulse rise and fall times 8-22
Pulse duty cycle 8-23
Settling time considerations 8-23
Measurement Settling Time Considerations 8-23
Reduction in gain-bandwidth 8-24
In this section: 9-1

9 System Expansion (TSP-Link) 9-1
Overview 9-2
Master and Slaves 9-2
System configurations 9-2

Connections 9-2
Initialization 9-3
Assigning node numbers 9-3
Resetting the TSP-Link 9-4
Using the expanded system 9-5
Accessing nodes 9-5
System behavior 9-5
Triggering with TSP-Link 9-6
TSP advanced features 9-6
Using groups to manage nodes on the TSP-Link network 9-9
Running parallel test scripts 9-10
Using the data queue for real-time communication 9-11
Copying test scripts across the TSP-Link network 9-12
Removing stale values from the reading buffer 9-12

10 Digital I/O and Triggering 10-1
Overview 10-2
Digital I/O port 10-2
Port configuration 10-2
Digital I/O configuration 10-3
Controlling digital I/O lines 10-4
Output Enable (Models 2601/2602) 10-7
Overview 10-7
Operation 10-7
Front panel control of Output Enable 10-8
Remote control of Output Enable 10-8
Interlock (Models 2612/2612/2635/2636) 10-8
Overview 10-8
Operation 10-8
TSP-Link Synchronization lines 10-10
Connecting to TSP-Link 10-10
Digital I/O 10-10
Remote TSP-Link synchronization line commands 10-10
Triggering 10-12
Triggering types 10-12
Measurement triggering 10-12
Front panel triggering 10-13
Remote triggering 10-14
Hardware trigger modes 10-15
Understanding synchronous triggering modes 10-20

11 Communications Interfaces 11-1
Overview 11-2
Selecting an interface 11-2
GPIB operation 11-2
GPIB standards 11-2
GPIB connections 11-2
Primary address 11-4
Terminator 11-5
General bus commands 11-6
REN (remote enable) 11-6
IFC (interface clear) 11-6
LLO (local lockout) 11-6
GTL (go to local) 11-6
DCL (device clear) 11-6
SDC (selective device clear) 11-7
GET (group execute trigger) 11-7
SPE, SPD (serial polling) 11-7
Front panel GPIB operation 11-7
Error and status messages 11-7
GPIB status indicators 11-7
LOCAL key 11-8
RS-232 interface operation 11-8

Setting RS-232 interface parameters 11-8
Sending and receiving data 11-9
Terminator 11-9
Baud rate 11-9
Data bits and parity 11-10
Flow control (signal handshaking) 11-10
RS-232 connections 11-11
Error messages 11-11

12 Instrument Control Library 12-1
Command programming notes 12-2
Conventions 12-2
Functions and attributes 12-3
TSP-Link nodes 12-4
Logical instruments 12-5
Reading buffers 12-5
Time and date values 12-7
ICL functions and attributes 12-8
data queue functions 12-15
delay function 12-17
digio functions and attributes 12-17
display functions and attributes 12-23
errorqueue functions and attribute 12-35
exit function 12-36
format attributes 12-36
gpib attribute 12-39
localnode attribute 12-39
makegetter functions 12-42
printbuffer and printnumber functions 12-43
reset function 12-45
serial functions and attributes 12-45
setup functions and attribute 12-48
smuX functions and attributes 12-48
status function and attributes 12-76
Status register sets 12-76
Status byte and SRQ 12-76
timer functions 12-108
trigger functions 12-109
tsplink function and attributes 12-110
userstring functions 12-116
waitcomplete function 12-118

13 Factory Scripts 13-1
Introduction 13-2
Factory script 13-2
KIGeneral 13-2
KIPulse 13-12
Advanced features for Models 2635 and 2636 13-13
Variable off time between pulses in a pulse train 13-13
Simultaneous IV measurement during pulse 13-13
Additional hardware triggering parameters 13-14
Flash firmware upgrade 13-35

14 Display Operations 14-1
Display functions and attributes 14-2
Display features 14-2
Display screen 14-2
Measurement functions 14-3
Display resolution 14-3
Display messages 14-4
Clearing the display 14-4
Cursor position 14-4
Displaying text messages 14-5
Input prompting 14-7

Menu 14-7
Parameter value prompting 14-8
Annunciators 14-9
LOCAL lockout 14-10
Load test menu 14-10
Saving a user script 14-11
Adding USER TESTS menu entries 14-11
Deleting USER TESTS menu entries 14-12
Running a test from the front panel 14-12
Display triggering 14-12
Key-press codes 14-13
Sending keycodes 14-13
Capturing key-press codes 14-13

15 Performance Verification 15-1
Introduction 15-2
Verification test requirements 15-2
Environmental conditions 15-2
Warm-up period 15-2
Line power 15-3
Recommended test equipment 15-3
Verification limits 15-3
Restoring factory defaults 15-4
Performing the verification test procedures 15-5
Test summary 15-5
Test considerations 15-5
Setting the source range and output value 15-5
Setting the measurement range 15-6
Output voltage accuracy 15-6
Voltage measurement accuracy 15-8
Output current accuracy 15-9
Series 2600 output current accuracy 100nA and higher 15-9
Model 2635/2636 output current accuracy 1nA to 100nA ranges 15-11
Current measurement accuracy 15-14
Series 2600 current measurement accuracy 100nA and higher 15-14
Model 2635/2636 current measurement accuracy 100pA to 100nA ranges 15-15

16 Calibration 16-1
Introduction 16-2
Environmental conditions 16-2
Temperature and relative humidity 16-2
Warm-up period 16-2
Line power 16-2
Calibration considerations 16-2
Calibration cycle 16-3
Recommended calibration equipment 16-3
Calibration errors 16-5
Calibration 16-5
Calibration steps 16-5
Calibration commands 16-7
Calibration procedure 16-8

17 Routine Maintenance 17-1
Introduction 17-2
Line fuse replacement 17-2
Front panel tests 17-3
Keys test 17-3
Display Patterns test 17-3

A Specifications A-1

B Error and Status Messages B-1
Introduction B-2
Error summary B-2
Error effects on scripts B-2
Reading errors B-2

C Common Commands C-1
Common commands C-2
Command summary C-2
Script command equivalents C-2
Command reference C-3

D Status Model D-1
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-8
Programming and reading registers D-9
Programming enable and transition registers D-9
Reading registers D-10
Status byte and service request (SRQ) D-10
Status byte register D-10
Service request enable register D-12
Serial polling and SRQ D-12
SPE, SPD (serial polling) D-12
Status byte and service request commands D-12
Enable and transition registers D-13
Controlling node and SRQ enable registers D-13
Status register sets D-15
System Summary Event Registers D-15
Standard Event Register D-16
Operation Event Registers D-18
Measurement Event Registers D-24
Register programming example D-27
Queues D-27
Output queue D-27
Error queue D-28
TSP-Link system status D-28
Status model configuration example D-28

E Speed Specification Test Conditions E-1
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-3
Sweep Operation Rates E-3
Digital I/O handshaking: E-4
Measure to Memory E-4
Measure to GPIB E-5
Source Measure to Memory E-5
Source Measure to GPIB E-5
Source Measure Pass/Fail to Memory E-5
Source Measure Pass/Fail to GPIB E-5
Single Measurement Rates E-5
Measure to GPIB E-6
Source Measure to GPIB E-6

Source Measure Pass/Fail to GPIB E-6
Function/ Range Change Rates E-6
Source Range Change Rate E-7
Measure Range Change Rate E-7
Function Change Rate E-7
Command Processing E-7

F Display Character Codes F-1
Introduction F-2
Display character dot patterns F-5

This page left blank intentionally.

List of Figures

Section	Figure	Title	Page
1	Figure 1-1	Models 2601, 2611, 2602, 2612, 2635, and 2636 front panels	1-6
1	Figure 1-2	Models 2601/2611 and 2602/2612 rear panels	1-9
1	Figure 1-3	Models 2635/2636 rear panels	1-11
1	Figure 1-4	Display modes	1-16
2	Figure 2-1	Script example	2-7
2	Figure 2-2	Programming model for scripts	2-9
2	Figure 2-3	GPIB cable	2-10
2	Figure 2-4	RS-232 cable (straight-through)	2-10
2	Figure 2-5	Test Script Builder (example)	2-12
2	Figure 2-6	Opening and closing communications	2-14
2	Figure 2-7	Creating and modifying a script using the Test Script Builder	2-15
2	Figure 2-8	Creating a project folder	2-16
2	Figure 2-9	Saving a script in the Test Script Builder	2-17
2	Figure 2-10	Creating a new script file	2-18
2	Figure 2-11	Renaming a project folder and/or script file	2-19
2	Figure 2-12	Changing a launch configuration	2-19
2	Figure 2-13	Opening the Run dialog box (launch configuration)	2-20
2	Figure 2-14	Run dialog box (Script Attributes tab)	2-22
2	Figure 2-15	Relaunching a script from the Test Script Builder toolbar	2-22
2	Figure 2-16	Re-launching a script from the Test Script Builder toolbar	2-23
2	Figure 2-17	Importing a script from memory of the Series 2600	2-24
2		Instrument Console icons	2-25
2	Figure 2-18	Programming interaction tabs: Problems, Tasks and Command Help	2-28
2	Figure 2-19	Programming interaction tabs: Language Help, Bookmarks, Browser View	2-29
2	Figure 2-20	Workspace Launcher and Select Workspace Directory	2-31
2	Figure 2-21	Importing a project from another workspace folder	2-32
2	Figure 2-22	Deleting a project	2-33
3	Figure 3-1	2602/2612 input/output connectors	3-3
3	Figure 3-2	Model 2636 input/output connectors	3-3
3	Figure 3-3	Model 2602/2612 input/output LO and chassis ground terminals	3-4
3	Figure 3-4	Model 2602/2612 Low-Noise Chassis Ground Banana Jack and Chassis Screw	3-5
3	Figure 3-5	Model 2602/2612 two-wire connections (local sensing)	3-6
3	Figure 3-6	Model 2602/2612 four-wire connections (remote sensing)	3-7
3	Figure 3-7	Contact check connections	3-9
3	Figure 3-8	Model 2602/2612 two SMUs connected to a 3-terminal device (local sensing)	3-10
3	Figure 3-9	Three SMUs connected to a 3-terminal device	3-11
3	Figure 3-10	Models 2602 and 2612 high-impedance guarding	3-12
3	Figure 3-11	Model 2636 high-impedance guarding (floating)	3-13
3	Figure 3-12	Model 2636 High-impedance guarding (non-floating)	3-13
3	Figure 3-13	Models 2602 and 2612 noise shield	3-14

3 Figure 3-14 Model 2636 noise shield (non-floating) 3-15
3 Figure 3-15 Model 2636 noise shield (floating) 3-15
3 Figure 3-16 Safety shield for hazardous voltage using two 2601/2602 channels (>42V) 3-16
3 Figure 3-17 Model 2601/2602-1 connections for test circuit shown in Figure 3-16 3-16
3 Figure 3-18 Safety shield for Models 2611/2612/2635/2636 hazardous voltage (200V maximum) 3-17
3 Figure 3-19 Model 2601/2602-1 connections for test circuit shown in Figure 3-18 3-17
3 Figure 3-20 Model 2636 connections for test circuit shown in Figure 3-18 3-18
3 Figure 3-21 Model 2601/2602-1 connections for noise shield, safety shield, and guarding 3-18
3 Figure 3-22 Model 2636 connections for noise shield, safety shield, and guarding 3-19
3 Figure 3-23 Floating the Series 2600 3-21
3 Figure 3-24 Model 2601/2602-1 SMU connections for the floating configuration shown in Figure 3-23 3-22
4 Figure 4-1 Fundamental source measure configuration 4-5
4 Figure 4-2 2-wire resistance sensing 4-14
4 Figure 4-3 4-wire resistance sensing 4-14
4 Figure 4-4 Contact check measurements 4-18
5 Figure 5-1 Comparison of staircase sweep types 5-3
5 Figure 5-2 Linear staircase sweep 5-4
5 Figure 5-3 Logarithmic staircase sweep (1V to 10V, five steps) 5-5
5 Figure 5-4 Pulse sweep example 5-6
5 Figure 5-5 Custom sweep example 5-7
6 Figure 6-1 Moving average and repeating filters 6-12
6 Figure 6-2 Media Filter 6-13
8 Figure 8-1 Two basic staircase sweep waveforms 8-3
8 Figure 8-2 Pulse sweep example 8-4
8 Figure 8-3 Model 2601/2602 continuous power operating boundaries 8-8
8 Figure 8-4 Model 2611/2612/2635/2636 continuous power operating boundaries 8-9
8 Figure 8-5 Model 2601/2602 I-Source boundaries 8-10
8 Figure 8-6 Model 2611/2612/2635/2636 I-Source boundaries 8-11
8 Figure 8-7 I-Source operating examples 8-12
8 Figure 8-8 Model 2601/2602 V-Source boundaries 8-13
8 Figure 8-9 Model 2611/2612/2635/2636 V-Source boundaries 8-14
8 Figure 8-10 V-Source operating examples 8-15
8 Figure 8-11 Source I configuration 8-16
8 Figure 8-12 Source V configuration 8-17
8 Figure 8-13 Measure only configurations 8-18
8 Figure 8-14 Contact check circuit configuration 8-19
8 Figure 8-15 Comparison of unguarded and guarded measurements 8-21
8 Figure 8-16 Pulse period 8-22
8 Figure 8-17 Pulse rise and fall times 8-22
9 Figure 9-1 TSP-Link connections 9-3
9 Figure 9-2 Multiple TSP-Link networks 9-7
9 Figure 9-3 Single TSP-Link network with groups 9-8
10 Figure 10-1 Digital I/O port 10-2
10 Figure 10-2 Digital I/O port configuration 10-4
10 Figure 10-3 Using Model 2601/2602 Output Enable 10-7
10 Figure 10-4 Using Model 2611/2612/2635/2636 Interlock 10-9
10 Figure 10-5 Measurement triggering sequence 10-13
11 Figure 11-1 IEEE-488 connector 11-3

11 Figure 11-2 IEEE-488 connections 11-3
11 Figure 11-3 IEEE-488 and RS-232 connector locations 11-4
11 Figure 11-4 RS-232 interface connector 11-10
13 Figure 13-1 Pulse sweep example 13-35
13 Figure 13-2 Pulse sweep example 13-35
14 Figure 14-1 Row/column format for display messaging 14-5
15 Figure 15-1 Connections for voltage verification 15-7
15 Figure 15-2 Current verification connections (2602/2612(3A); 2636(1.5A)) .. 15-12
15 Figure 15-3 Current verification connection ranges (2601/2602 (3A); 2611/2612/2635/2636 (1.5A)) 15-13
16 Figure 16-1 Connections for voltage calibration 16-9
16 Figure 16-2 Connections for current calibration (100nA to 1A ranges) 16-13
16 Figure 16-3 Connections for current calibration 16-17
16 Figure 16-4 Connections for contact check 0W calibration 16-19
16 Figure 16-5 Connections for contact check 50W calibration 16-20
17 Figure 17-1 Line fuse replacement 17-2
D Figure D-1 Status model overview D-3
D Figure D-2 Status model (system summary and standard event registers) D-4
D Figure D-3 Status model (operation event registers) D-5
D Figure D-4 Status model (questionable event registers) D-6
D Figure D-5 Status model (measurement event registers) D-7
D Figure D-6 16-bit status register D-9
D Figure D-7 Status byte and service request (SRQ) D-11
D Figure D-8 Standard event register D-17
D Figure D-9 TSP-Link status model configuration example D-30

This page left blank intentionally.

List of Tables

Section	Table	Title	Page
1	Table 1-1	Connectors and triax cable conductors	1-12
1	Table 1-2	Triax connector on ground module	1-13
1	Table 1-3	Main Menu	1-19
1	Table 1-4	Configuration menus	1-20
1	Table 1-5	Default settings	1-23
2	Table 2-1	Example script to sweep V and measure I	2-34
2	Table 2-2	Example script using a function	2-35
2	Table 2-3	Example interactive chunk fragment for a script	2-36
3	Table 3-1	Selecting the sense mode from the front panel	3-8
3	Table 3-2	Commands to select sense mode	3-8
3	Table 3-3	Commands to select output-off state	3-24
4	Table 4-1	Source-measure capabilities	4-3
4	Table 4-2	Maximum compliance values	4-4
4	Table 4-3	Compliance commands	4-5
4	Table 4-4	Auto zero settings	4-6
4	Table 4-5	Auto zero command and options	4-7
4	Table 4-6	Basic source-measure commands	4-10
4	Table 4-7	Basic contact check commands	4-18
5	Table 5-1	Logarithmic sweep points	5-5
5	Table 5-2	Staircase sweep functions	5-8
5	Table 5-3	Pulse sweep functions	5-8
5	Table 5-4	Custom sweep functions	5-9
5	Table 5-5	Sweep example parameters	5-9
6	Table 6-1	Source and measurement ranges	6-2
6	Table 6-2	Range commands	6-5
6	Table 6-3	Digits commands	6-6
6	Table 6-4	Speed command	6-7
6	Table 6-5	Rel commands	6-9
6	Table 6-6	Filter commands	6-13
7	Table 7-1	Data store commands	7-4
7	Table 7-2	Buffer storage control attributes	7-6
7	Table 7-3	Buffer read-only attributes	7-6
7	Table 7-4	Buffer control programming examples	7-6
7	Table 7-5	Buffer read-only attribute programming examples	7-6
7	Table 7-6	Recall attributes	7-7
7	Table 7-7	Buffer status bits	7-8
8	Table 8-1	Maximum compliance limits	8-2
8	Table 8-2	Model 2601/2602 Maximum Duty Cycle equation constants	8-6
8	Table 8-3	Model 2611/2612/2635/2636 Maximum Duty Cycle equation constants	8-6
8	Table 8-4	Current Measure Settling Time1, 2	8-23
8	Table 8-5	Current source gain-bandwidth	8-24
9	Table 9-1	TSP-Link reset commands	9-4
9	Table 9-2	TSP-Link network group functions	9-9

10 Table 10-1 Digital I/O bit weighting 10-5
10 Table 10-2 Digital I/O commands..... 10-6
10 Table 10-3 Digital I/O bit weighting 10-10
10 Table 10-4 TSP-Link Triggering commands..... 10-10
10 Table 10-5 Hardware trigger mode summary 10-15
11 Table 11-1 General bus commands..... 11-6
11 Table 11-2 RS-232 interface commands 11-9
11 Table 11-3 RS-232 connector pinout 11-10
11 Table 11-4 PC serial port pinout 11-11
13 Table 13-1 KIGeneral TSP test script: PulseIMeasureV 13-2
13 Table 13-2 KIGeneral TSP test script: PulseVMeasureI 13-3
13 Table 13-3 KIGeneral TSP test script: SweepILinMeasureV 13-4
13 Table 13-4 KIGeneral TSP test script: SweepVLinMeasureI 13-5
13 Table 13-5 KIGeneral TSP test script: SweepILogMeasureV 13-7
13 Table 13-6 KIGeneral TSP test script: SweepVLogMeasureI 13-8
13 Table 13-7 KIGeneral TSP test script: SweepIListMeasureV 13-10
13 Table 13-8 KIGeneral TSP test script: SweepVListMeasureI 13-11
13 Table 13-9 Required true conditions for function execution..... 13-12
13 Table 13-10 KIGeneral TSP test script: ConfigPulseIMeasureV..... 13-14
13 Table 13-11 KIGeneral TSP test script: ConfigPulseVMeasureI..... 13-16
13 Table 13-12 KIGeneral TSP test script: ConfigPulseIMeasureVSweepLin.... 13-18
13 Table 13-13 KIGeneral TSP test script: ConfigPulseVMeasureISweepLin.... 13-20
13 Table 13-14 KIGeneral TSP test script: ConfigPulseIMeasureVSweepLog... 13-22
13 Table 13-15 KIGeneral TSP test script: ConfigPulseIMeasureVSweepLog... 13-24
13 Table 13-16 KIGeneral TSP test script: ConfigPulseVMeasureISweepLog... 13-26
13 Table 13-17 KIGeneral TSP test script: QueryPulseConfig 13-28
13 Table 13-18 KIGeneral TSP test script: InitiatePulseTest 13-30
13 Table 13-19 KIGeneral TSP test script: InitiatePulseTestDual..... 13-32
14 Table 14-1 Cross referencing functions/attributes to section topics..... 14-2
14 Table 14-2 Bit identification for annunciators 14-10
14 Table 14-3 Keycodes to send for display.sendkey 14-13
14 Table 14-4 Keycode values returned for display.getlastkey 14-14
15 Table 15-1 Recommended verification equipment..... 15-3
15 Table 15-2 Model 2601/2602 output voltage accuracy limits 15-8
15 Table 15-3 Model 2611/2612/2635/2636 output voltage accuracy limits 15-8
15 Table 15-4 Model 2601/2602 voltage measurement accuracy limits 15-9
15 Table 15-5 Model 2611/2612/2635/2636 voltage measurement accuracy limits 15-9
15 Table 15-6 Model 2601/2602 output current accuracy limits 15-10
15 Table 15-7 Model 2611/2612 output current accuracy limits..... 15-10
15 Table 15-8 Model 2635/2636 output current accuracy limits 15-14
15 Table 15-9 Model 2635/2636 Characterization of Voltage Source settings . 15-15
15 Table 15-10 Model 2601/2602 current measurement accuracy limits 15-15
15 Table 15-11 Model 2611/2612 current measurement accuracy limits..... 15-16
15 Table 15-12 Model 2635/2636 current measurement accuracy limits 15-16
16 Table 16-1 Recommended calibration equipment 16-4
16 Table 16-2 Model 2601/2602 calibration steps 16-5
16 Table 16-3 Model 2611/2612 calibration steps 16-6
16 Table 16-4 Model 2635/2636 calibration steps 16-6
16 Table 16-5 Calibration commands 16-7
16 Table 16-6 Settings of Model 2635/2636 Characterization of Voltage Source..... 16-18
17 Table 17-1 Line fuse 17-3
B Table B-1 Error queue commands B-2
B Table B-2 Error summary B-3

C	Table C-1	Common commands	C-2
C	Table C-2	Script command equivalents	C-2
D	Table D-1	Status functions and registers	D-8
D	Table D-2	Commands to reset registers and clear queues	D-9
D	Table D-3	Status Byte and Service Request Enable Register commands	D-13
D	Table D-4	System node and SRQ enable register bit attributes	D-14
D	Table D-5	Standard event commands	D-17
D	Table D-7	Operation event commands	D-18
D	Table D-6	Status event status registers and bits	D-18
D	Table D-8	Operation event commands	D-23
D	Table D-9	Operation event commands	D-25
D	Table D-10	Error queue commands	D-28
F	Table F-1	Display character codes (decimal 0-143)	F-2
F	Table F-2	Display character codes (decimal 144-255)	F-4

This page left blank intentionally.

Section 1

Getting Started

In this section:

Topic	Page
Introduction	**1-2**
Capabilities and features	1-2
Organization of manual sections	1-3
General information	**1-3**
Warranty information	1-3
Contact information	1-3
Safety symbols and terms	1-3
Unpacking and inspection	1-3
Options and accessories	1-4
User's and Reference manuals	1-5
Front and rear panel familiarization	**1-6**
Front panel summaries	1-6
Rear panel summaries	1-9
Cooling vents	**1-13**
Power-up	**1-14**
Line power connection	1-14
Power-up sequence	1-15
Beeper	1-15
Display modes	**1-16**
Editing controls	**1-17**
Source and compliance editing	1-17
Menu navigation	1-18
Menu types	1-19
Interface selection	**1-20**
To select the GPIB interface	1-21
To select the RS-232 interface	1-21
Error and status messages	**1-21**
Default settings	**1-21**
Front panel setups	1-21
Remote operation setups	1-22
Remote programming	**1-24**
Requesting readings	1-24
Requesting command settings	1-24

Introduction

Keithley Instruments Series 2600 System SourceMeter® instruments offer electronic component and semiconductor device manufacturers a scalable, high throughput, highly cost-effective solution for precision DC, pulse, and low frequency AC source-measure testing.

Capabilities and features

- Models 2601/2602 System SourceMeters:
  - Source ±DC voltage from 1µV to 40.4V
  - Source ±DC current from 1pA to 3.03A
  - Measure ±DC voltage from 1µV to 40.8V
  - Measure ±DC current from 1pA to 3.06A
- Models 2611/2612 System SourceMeters:
  - Source ±DC voltage from 1µV to 202V
  - Source ±DC current from 1pA to 1.515A
  - Measure ±DC voltage from 1µV to 204V
  - Measure ±DC current from 1pA to 1.53A
- Models 2635/2636 System SourceMeters:
  - Source +/- DC voltage from 1uV to 202V
  - Source +/- DC current from 20fA to 1.515A
  - Measure +/- DC voltage from 1uV to 204V
  - Measure +/- DC current from 1fA to 1.53A
- Resistance and power measurement functions
- Contact check function[1]
- Two independent SourceMeter channels (Models 2602, 2612, and 2636 only)
- Four-quadrant sink or source operation
- 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 Series 2600 to control other devices

1. All Model 2611/2612 System SourceMeters manufactured by Keithley Instruments support the contact check function. Models 2635 and 2636 do not support the contact check function. Only Models 2601/2602 with firmware Revision 1.1.0 or later and source measure unit (SMU) hardware Revision E or later support the contact check function. To determine the firmware and SMU hardware revisions, inspect the data returned by the `print(localnode.info())` command. The **InstFwRev** and **SMUBrdRev** keys contain the necessary information.

Organization of manual sections

The manual sections in the PDF version of this manual 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 Series 2600 require warranty service, contact the Keithley Instruments representative or authorized repair facility in your area for further information. When returning the instrument for repair, be sure to complete and return the service form at the back of this manual to provide the repair facility with the relevant information.

Contact information

If you have any questions, please contact your local Keithley Instruments representative or call one of our Application Engineers at 1-888-KEITHLEY (1-888-534-8453), U.S. and Canada only. You can also contact us through our website at 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 Series 2600 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 Series 2600 from the bag, observe the following handling precautions.

Handling precautions

- Always grasp the Series 2600 by the covers or by the handle.
- After removing the Series 2600 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 Series 2600 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 Series 2600 order:

- Model 2601, 2602, 2611, 2612, 2635, or 2636 SourceMeter with line cord
- TSP-Link cable for Models 2601, 2602, 2611, and 2612
- Accessories as ordered
- Certificate of calibration
- CD-ROMs that contain:
  - PDFs of the User's and Reference Manuals
  - Test Script Builder script development software

The following additional items are included with the Models 2611, 2612, 2635, and 2636 SourceMeters:

- Model 2600-IAC Interlock connector adaptor

The following item is for Models 2601, 2602, 2611, and 2612 only:

- Device under test (DUT) interface connector kit for each SourceMeter channel; kit includes one hooded screw terminal connector that mates with the SourceMeter measurement terminals

The following items are for Models 2635 and 2636 only:

- SC-73-0 6" ground wire.
- TSP-Link cable
- 2600-ALG-2 low noise triax cable with alligator clips, UL approved for up to 42V, 2m (6.6ft) (2 with Model 2636 and 1 with Model 2635)
- Quick start guide

Options and accessories

GPIB cables, interfaces, and adaptors (connects Series 2600 to the GPIB bus)

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

KPCI-488LP IEEE-488: Interface/Controller for the PCI Bus.

KPXI-488 IEEE-488: Interface Board for the PXI Bus.

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.

KUSB-488A IEEE-488: USB-to-GPIB Interface Adaptor

RS-232 cable (connects Series 2600 to the RS-232)

Model 7009-5 shielded RS-232 cable: This straight-through cable connects the RS-232 of the Series 2600 to the RS-232 interface of your PC. This cable is 5ft. long and uses shielded cable and connectors to reduce electromagnetic interference (EMI).

TSP-Link cable (connects Series 2600 to the TSP-Link)

CA-180-3A CAT 5 cable: This crossover CAT5 LAN cable connects the TSP-Link of the Series 2600 to the TSP-Link of other instruments.

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

Series 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 and Reference manuals

The User's and Reference Manuals are provided on the product information CD-ROM in PDF format. The User's Manual provides the fundamental operating information for the instrument. The Reference Manual provides additional information on the topics covered in the User's Manual. The Reference Manual also includes advanced operation topics and maintenance information.

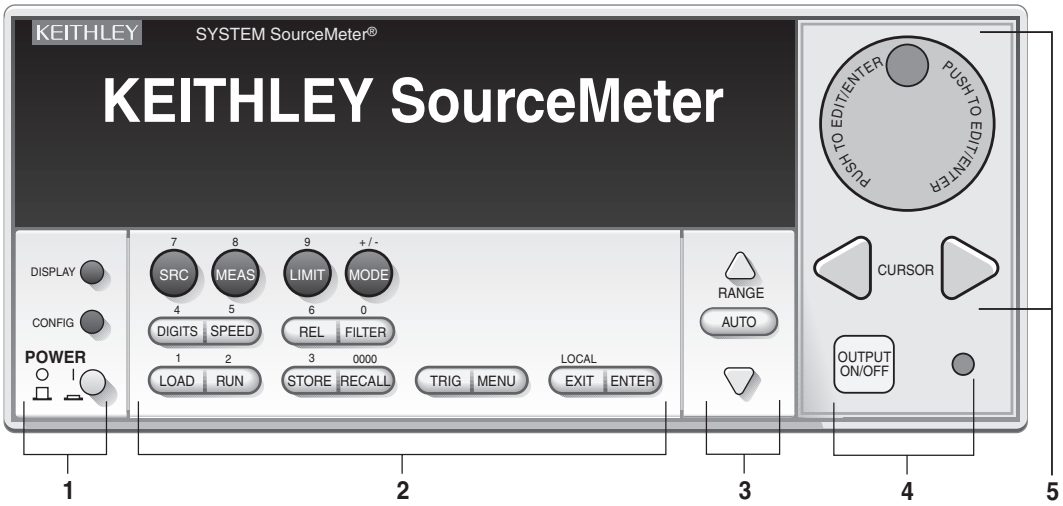
Front and rear panel familiarization

Front panel summaries

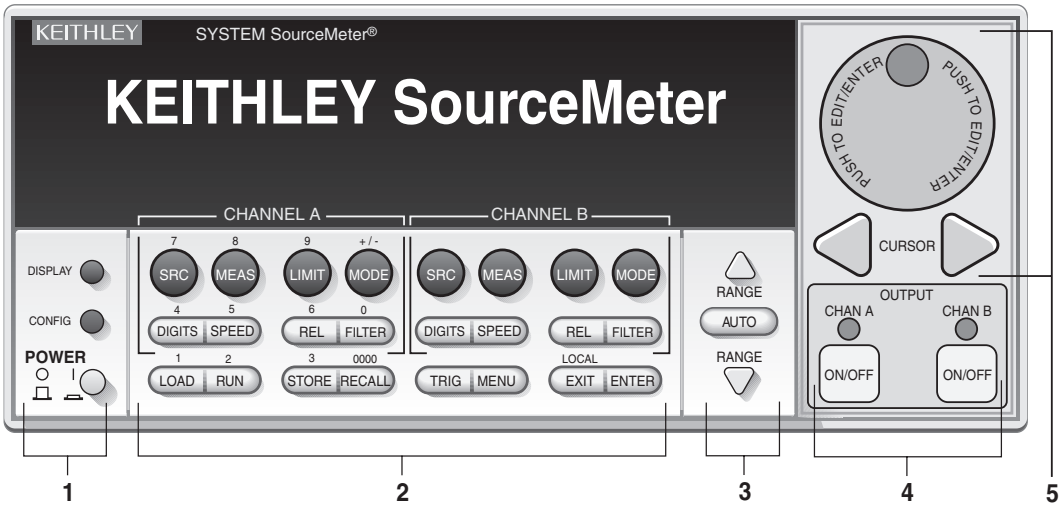
The front panels of the Series 2600 are shown in Figure 1-1. The descriptions of the front panel controls follow Figure 1-1.

Figure 1-1
Models 2601, 2611, 2602, 2612, 2635, and 2636 front panels

Models 2601, 2611, and 2635



Models 2602, 2612, and 2636



NOTE The Models 2601, 2611, and 2635 have one SourceMeter channel (Channel A), and the Models 2602, 2612, and 2636 have 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 2602/2612/2636 single or dual-channel display.
CONFIG	Use to configure a function or operation.
POWER	Power switch: The in position turns SourceMeter on (I); the 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

Models 2601, 2602, 2611, 2612, 2635, and 2636:

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, Ω, 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, Ω, or W).

Models 2602, 2612, and 2636 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, Ω, 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, Ω, or W).

Middle row

Models 2601, 2602, 2611, 2612, 2635, and 2636:

DIGITS	Channel A changes display resolution 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.

Models 2602, 2612, and 2636 only:

DIGITS	Channel B changes display resolution 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 the last selected factory or user-defined scripts.
STORE	Stores readings, source values, and timestamp values in one of two internal buffers per channel 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 a remote interface, line frequency, self-tests, serial number, and beeper control.
EXIT	Cancels selection and backs out of menu structure. Also used as a **LOCAL** key to take the unit out of remote.
ENTER	Accepts selection and moves to the next choice or exits the menu.

3 Range keys:

△ **and** ▽	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 the **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.

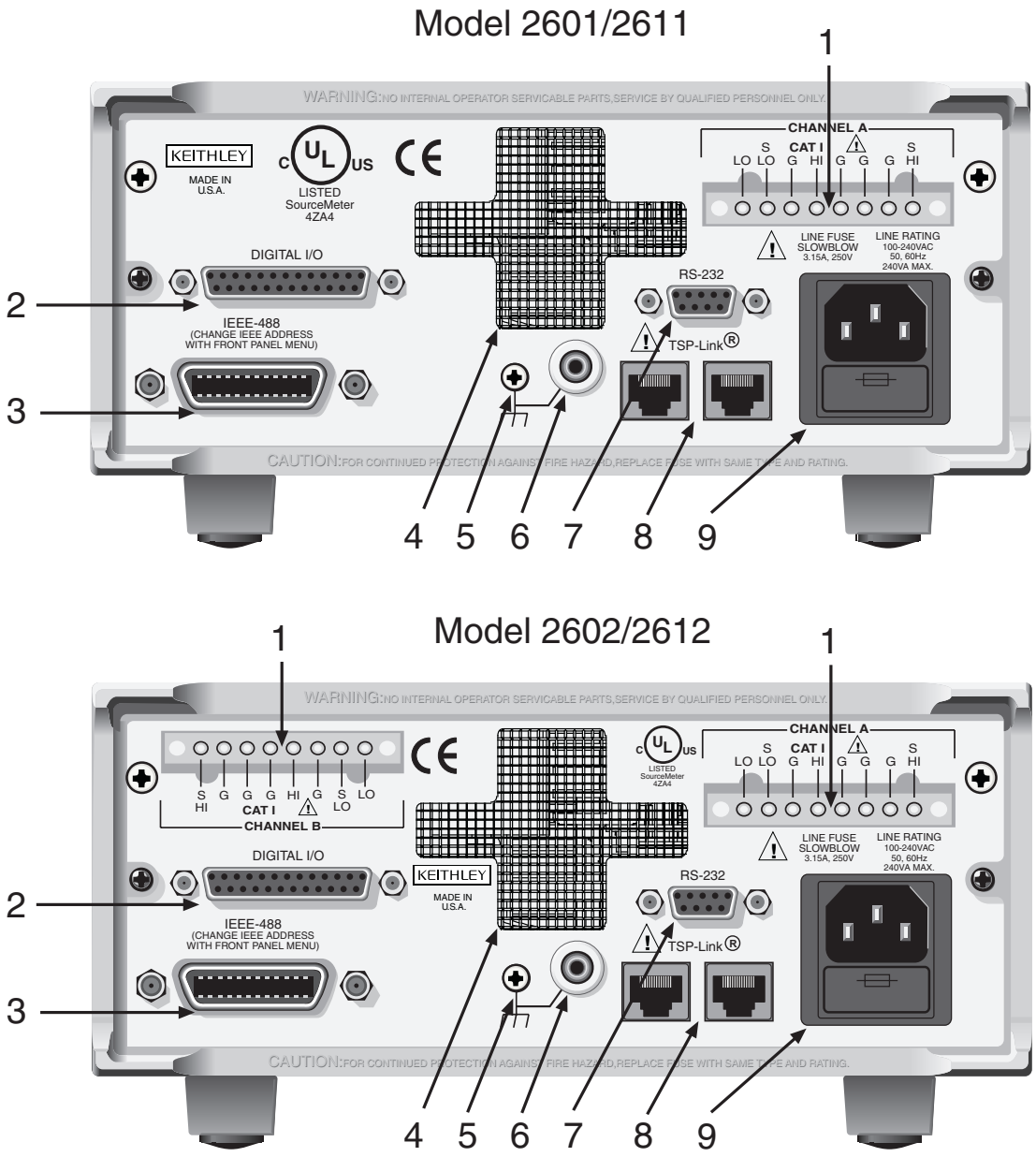
6 Display annunciators (not shown):

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

Rear panel summaries

The rear panels of Models 2601/2611 and Models 2602/2612 are shown in Figure 1-2. The descriptions of the rear panel components follow Figure 1-2. The rear panels of Models 2625 and 2636 are shown in Figure 1-3. The descriptions of the rear panel components follow Figure 1-3.

Figure 1-2
Models 2601/2611 and 2602/2612 rear panels



1 CHANNEL A and CHANNEL B (Channel B on 2602/2612 only)

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

2 DIGITAL I/O

Female DB-25 connector. Fourteen pins for digital input or output, one pin for output enable (2601/2602) or safety interlock (2611/2612); +5V and GND pins are also provided.

Use a cable equipped with a male DB-25 connector (Keithley Instruments 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 the internal cooling fan. Keep the 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 Instruments Model 7009-5) for connection to the PC.

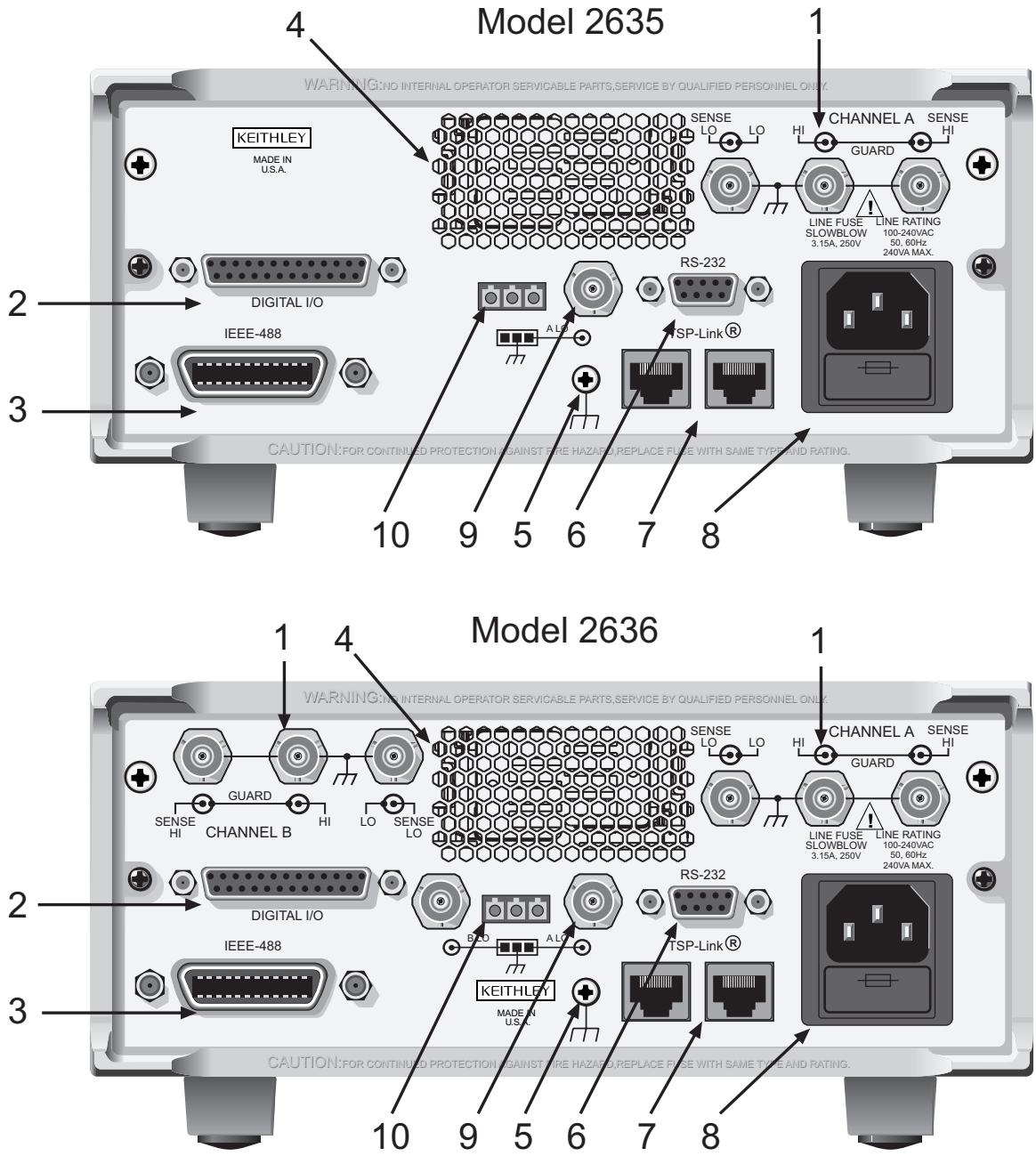
8 TSP-Link

Expansion interface that allows a Series 2600 and other TSP-enabled instruments to trigger and communicate with each other. Use a category 5e or higher LAN crossover cable (Keithley Instruments part number CA-180-3A).

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 50Hz or 60Hz.

Figure 1-3
Models 2635/2636 rear panels



1. CHANNEL A and CHANNEL B (Channel B on Model 2636 only)

Triax connectors for input/output, guard, and sense connections. Use only low-noise triax cables such as the Keithley Instruments Model 7078-TRX (available in several lengths). Connector terminals and associated triax cable conductors are as follows:

Table 1-1
Connectors and triax cable conductors

Connector	Center conductor	Inner ring	Outer ring
LO	Sense LO	Input/Output LO	Chassis ground
HI	Input/Output HI	Guard	Chassis ground
SENSE HI	Sense HI	Guard	Chassis ground
Triax cable	Center conductor	Inner shield	Outer shield

WARNING ***When connecting to the model 2611, 2612, 2635 and 2636 SMU outputs, with cables not rated for voltages above 42V, such as the 2600-ALG-2, you must disable the high voltage output by using the INTERLOCK function as defined in section 10 of this manual. Leaving the high voltage enabled while not properly insulating the external connections to the unit poses a shock hazard which could cause serious injury to the user. It is also recommended that the LO connection terminal not be allowed to float by connecting it to signal ground or another known signal reference.***

2. DIGITAL I/O

Female DB-25 connector. Fourteen pins for digital input or output, one pin for safety interlock. Use a cable equipped with a male DB-25 connector (Keithley Instruments 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. RS-232

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

7. TSP-Link

Expansion interface that allows a Series 2600 and other TSP-enabled instruments to trigger and communicate with each other. Use a category 5e or higher LAN crossover cable (Keithley Instruments part number CA-180-3A).

8. 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 50Hz or 60Hz. See Section 17 of this manual for line fuse replacement instructions.

9. Triax connector on Ground Module

Channel A and Channel B low noise chassis ground triax connectors. Use only low-noise triax cables such as the Keithley Model 7078-TRX. Connector terminals and associated triax cable connectors are as follows:

Table 1-2
Triax connector on ground module

Connector	Center conductor	Inner ring	Outer ring
LO	Output Lo	Floating	Chassis Ground
Triax cable	Center conductor	Inner shield	Outer shield

10. Phoenix connector on Ground Module

Channel A and Channel B Low noise chassis ground Phoenix connector.

Cooling vents

The Series 2600 has side intake and rear exhaust vents. One side must be unobstructed when rack mounted to dissipate heat. NEVER place a container of liquid (water or coffee for instance) 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 Series 2600 and degrade its performance. The Series 2600 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.

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 Series 2600 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 Series 2600, 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 Series 2600 surfaces to remain within specified limits under all operating conditions.

Rack mounting high power dissipation equipment adjacent to the Series 2600 could cause excessive heating to occur. The specified ambient temperature must be maintained around the surfaces of the Series 2600 to specified accuracies. A good measure to ensure proper cooling in rack situations with convection cooling only is to place the hottest equipment (for instance, the power supply) at the top of the rack. Precision equipment (such as the Series 2600) should be placed as low as possible in the rack where temperatures are coolest. Adding space panels below the Series 2600 will help ensure adequate air flow.

Power-up

Line power connection

Follow the procedure below to connect the Series 2600 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.
3. Connect the other end of the power cord to a grounded AC outlet.

WARNING	***The power cord supplied with the Series 2600 contains a separate ground wire for use with grounded outlets. When proper connections are made, the instrument chassis is connected to power line ground through the ground wire in the power cord. Failure to use a grounded outlet may result in personal injury or death due to electric shock.***

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

Line frequency

The Series 2600 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** > **LINE-FREQ** and then press **ENTER**.
2. Select the appropriate frequency and then press **ENTER**.
   Note: Select AUTO to automatically detected the line frequency.
3. 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 (refer to Figure 1-2 for Models 2601/2602/2611/2612 and Figure 1-3 for Models 2635/2636). This fuse protects the power line input of the instrument. If the line voltage fuse needs to be replaced, refer to "Line fuse replacement" in Section 17.

Power-up sequence

On power-up, the Series 2600 performs self-tests on its ROM and NVRAM 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 Series 2600 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:

1. KEITHLEY MODEL 26xx[1]
2. NODE = 1 GPIB = 26

- The node and serial port parameters are displayed briefly:

3. KEITHLEY MODEL 26xx[2]
4. NODE = 1 SERIAL = 9600,8,N,1,NONE

- If the line frequency setting is AUTO, a screen will be displayed indicating the power line frequency is being detected.

System identification

Serial number, firmware revision, and calibration dates can be displayed by selecting the SERIAL# item of the main menu.

Complete the following steps to view the system information.

1. Press **MENU** > **SYSTEM-INFO**.
2. Choose one of the following:
   - FIRMWARE
   - SERIAL#
   - CAL

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.

Complete the following steps to turn the beeper on or off.

1. select **MENU** > **BEEPER**.
2. Choose one of the following:

1. Unit displays actual model number: 2601, 2602, 2611, 2612, 2635, or 2636.
2. Unit displays actual model number: 2601, 2602, 2611, 2612, 2635, or 2636.

- **ENABLE**
- **DISABLE**

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

```
beeper.enable = 1
```

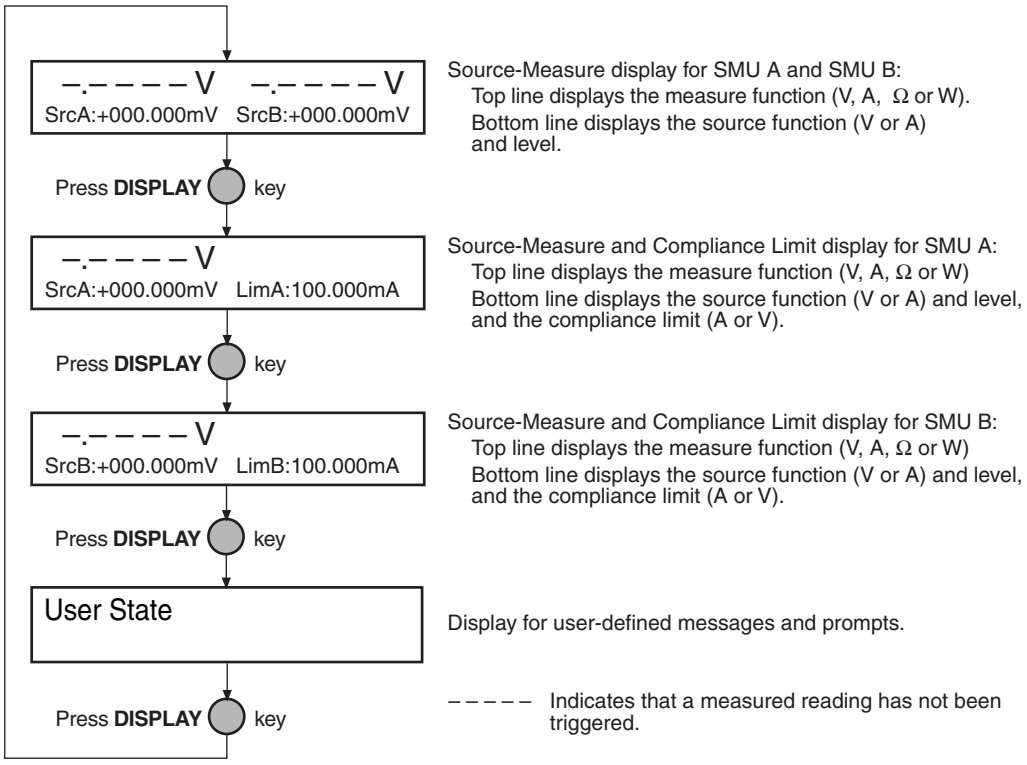
Display modes

Use the **DISPLAY** key to cycle through the various display modes shown in Figure 1-4.

(Models 2602, 2612, and 2636 only) Press the Display key more than once to cycle through the dual channel and single channel display modes. This applies to CHANNEL A (SMU A) and CHANNEL B (SMU B).

The Models 2601, 2611, and 2635 are a single channel (SMU A). The User State display messages are defined with specific display commands (refer to Section 14 for more information on display messaging).

Figure 1-4
Display modes



Editing controls

Source and compliance editing

When the Series 2600 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

Complete the following steps to edit the source.

1. Press the **SRC** key.
   The cursor flashes in the source value field.
2. Use the **CURSOR** arrow keys to move the cursor to the desired digit using.
3. Push the **Wheel** in to edit the source value.
   The Edit indicator displays.
4. Do one of the following to change the source value:
   - Rotate the wheel to adjust the digit.
     Note: The digit is automatically overflow or underflow to the next digit if the maximum source value is reached.
   - Use the numeric keys (0-9, +/-, 0000) to enter the source value.
     Note: The +/- toggles the polarity, and 0000 sets the value to 0.
5. Once the desired value displays, press **ENTER**.
   Note: The Edit indicator does not display.
6. (Optional) Press the **EXIT** key to cancel source editing.

Editing compliance values

Complete the following steps to edit the compliance value.

1. Do one of the following:
   - (Model 2601/2611/2635 and 2602/2612/2636 in single-channel display mode only) Press the **LIMIT** key.
   - (Model 2602/2612/2636 dual-channel display mode only) Press **LIMIT** or **CONFIG,** then **LIMIT** to edit the compliance limit.
2. Choose one of the following:
   - **VOLTAGE**
   - **CURRENT**
3. Use the **CURSOR** arrow keys to move the cursor to the desired value.
4. Press in on the wheel to enter edit mode.
   The Edit indicator displays.
5. Do one of the following to modify the compliance limit value:
   - Rotate the wheel to adjust the value.
     Note: The value is automatically overflow or underflow to the next value if the minimum or maximum value is reached.
   - Use the numeric keys (0-9) to enter the value.
6. Press **ENTER** to complete editing.
   Note: The Edit indicator does not display.
7. (Optional) Press the **EXIT** key cancel changes.

Menu navigation

When the Series 2600 is not in the edit mode (the Edit indicator does not display), the editing controls are used to navigate the Main and Configuration menus (refer to the "menus" topic later in Section 1) 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 arrow keys to select a menu or an option.
2. Press the ENTER key to select an item or menu option.
3. Rotate the wheel (clockwise or counter-clockwise) to select a value.

NOTE You can use the wheel to select items from the menu or a submenu.

4. Use the **EXIT** key to cancel changes or to return to the main menu.

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** arrow keys to move the cursor to the value that you want to edited.
2. Push in on the wheel to enter edit mode.
   The Edit indicator displays.
3. Rotate the wheel to set the appropriate value.

NOTE Adjusting past the maximum or minimum value automatically moves the cursor to the next higher or lower value for editing.

4. Press **ENTER** to select the value. Press **EXIT** to cancel the change.

Numeric entry method

1. Use the **CURSOR** arrow keys to move the cursor to the value that you want to edit.
2. Press the number entry key (**0** to **9**).
   The cursor moves to the next value on the right.
3. Repeat Step 2. as required to set the desired values.
4. Press **ENTER** to select the value.
5. (Optional) Press **EXIT** the cancel change and to return to the main menu.

Menu types

Many aspects of operation are configured through menus. There are two types of menus. Refer to the "Menu navigation" topic in this section for more details on using menus.

Main menu

The main menu is summarized in Table 1-3, along with the reference for each main selection.

To access the menu items shown in Table 1-3, press the **MENU** key, and then make your selection.

NOTE Use the menu options in Table 1-3 with v1.4.0 or higher.

Table 1-3
Main menu

Menu selection	Description	Reference
Setup	Saves and recalls user and factory setup options.	Section 1
SAVE	Saves user setup options.	
RECALL	Recalls user setup options.	
POWERON	Sets the default configuration.	
GPIB	Configure the GPIB interface options.	Section 11
ADDRESS	Configure the address for the GIPB interface.	Section 11
ENABLE	Enables and disables the GPIB interface.	Section 9
RS232	Controls the options for the RS232 interface.	Section 11
ENABLE	Use to enable and disable the RS232 interface.	
BAUD	Set the baud rate.	
BITS	Configure the number of bits.	
PARITY	Set the parity.	
FLOW-CTRL		
TSPLINK	An alternate way to configure the instrument for TSP-Link.	Section 9
NODE	Selects the instrument node identifier.	
RESET	Use to reset the TSP-link network.	
DISPLAY	Use to perform the display tests.	Section 17
NUMPAD	Enables and disables the NUMPAD.	
TEST	Runs the display test.	
DIGOUT	Preforms display test patterns.	Section 10
DIG-IO-OUTPUT		
WRITE-PROTECT		
BEEPER	Controls the key beeps.	Section 1
ENABLE	Enables the key beeps.	
DISABLE	Disables key beeps.	

Menu selection	Description	Reference
LINE-FREQ	Configure the line frequency.	Section 1
AUTO	Automatically selects the line frequency.	
50Hz		
60Hz		
SYSTEM-INFO	Displays the system information.	Section 1
FIRMWARE	Displays the version of firmware installed.	
SERIAL#	Displays the serial number of the unit.	
CAL	Displays the last calibration date.	

Configuration menus

The configuration menus are summarized in Table 1-4, 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 1-4
Configuration menus

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

Interface selection

The following summarizes basic interface selection for the Series 2600. Details on the interfaces, including configuration, are provided in Section 11. Use the editing controls for "Menu navigation" described earlier in this section to select and configure the interface.

To select the GPIB interface

1. Press **MENU** > **GPIB** and then press **ENTER**.
2. Choose **ADDRESS**, then press **ENTER**.
3. Set the **GPIB** address (0 to 30) and press **ENTER**.
4. Press **EXIT** to return to the main menu.

To select the RS-232 interface

1. Press **MENU** > **RS-232**, then press **ENTER**.
2. Do the following:
   - 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.
3. Press **EXIT** to return to the main menu.

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 notifications, as listed in Appendix B.

Messages, both status and error, are held in queues. For information on retrieving error messages from queues, refer to Appendix D.

Default settings

The Series 2600 can be restored to one of six setup configurations: Five user-saved setups, and the original factory defaults. As shipped from the factory, the Series 2600 powers-up to original default settings, which are also saved in the five user setup locations. Original default settings are listed in Table 1-5. 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 Series 2600 for the desired operating modes to be saved.
2. Press **MENU** > **SETUP** and then press **ENTER**.
3. Select **SAVE** menu item, then press **ENTER**.
4. Select the user number (1 through 5), and press **ENTER**.

To restore a setup:

1. Press the **MENU** key to access the main menu.
2. Select **SETUP**, then press **ENTER**.
3. Select the **RECALL** menu item, then press **ENTER**.
4. Do one of the following:
   - Select the user number (1 through 5), and then press **ENTER**

- Select **FACTORY** to restore factory defaults, and press **ENTER**.

To select power-on setup:

1. Press the **MENU** > **SETUP** > **POWERON**, and then press **ENTER**
2. Do one of the following:
   - Choose **FACTORY** to load the original defaults
   - Select **USER NUMBER** (1 through 5) to load a user preference.
3. Press **ENTER**.
4. Press **EXIT** to return to the main menu.

Remote operation setups

To save and recall 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.

This page left blank intentionally.

where:
n = 1, 2, 3, 4 or 5

To restore default setups:

The reset commands return the Series 2600 to the original factory defaults:

`reset()`	Restore all factory defaults.
`smua.reset()`	Restore Channel A defaults.
`smub.reset()`	Restore 2602/2612/2636 Channel B defaults.

To select 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.

where:
n = 0 (*RST defaults)
n = 1 to 5 (user setups 1-5)

Table 1-5
Default settings

Setting	Default
A/D controls:	
Auto-zero	Auto
Line frequency	Auto (firmware version 1.2.0 or higher) No effect (firmware versions lower than 1.2.0)
Beeper	On
Data store	No effect
Digital output:	
Output value	No effect
Write protect	No effect
Digits	5-1/2
Display mode (Models 2602/2612/2636)	Dual-channel
Filter:	Off
Averaging type	Repeat
Count	1
GPIB address	No effect
Limit value:	
Current limit	1A Models (2601/2602/2611/2612) 100mA Models (2635/2636)
Voltage limit	20V Models (2635/2636) 40V Models (2601/2602) 200V Models (2611/2612)
Measure:	
Function	Voltage
I-meter range	100mA
V-meter range	100mV Models (2601/2602) 200mV Models (2611/2612/2635/2636)
Output	Off
Off state	Normal
Rel	Off
Current value	0.0pA
Voltage value	0mV
Ohms value	0mΩ
Watts value	0mW
RS-232	No effect
Setting	**Default**

Table 1-5 (continued)
Default settings

Sense mode	2-wire
Source:	
Function	Voltage
Current value	0A
Voltage value	0V
Current range	100nA Models (2601/2602/2611/2612) 1nA Models (2635/2636)
Voltage range	100mV Models (2601/2602) 200mV Models /2611/2612/2635/2636)
Speed	Normal (1 PLC)
Triggering:	
Trigger-in source	Immediate
Count	Finite
Interval	0s
Delay	0s
TSP-Link node	1

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 Section 2, Section 12, and Section 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)
```

Section 2

TSP Programming

In this section:

Topic	Page
Introduction	**2-3**
Test Script Processor (TSP)	2-3
Run-time environment	2-3
Queries	2-4
Scripts	2-4
Named scripts	2-4
Functions	2-5
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	2-7
TSP programming levels	2-8
Programming model for scripts	2-8
Installing the Test Script Builder software	**2-9**
System connections	**2-9**
GPIB	2-9
RS-232	2-10
Using Test Script Builder	**2-11**
Project Navigator	2-11
Script Editor	2-11
Programming Interaction	2-11
Creating and modifying a script	2-15
Script launch configuration	2-19
Launching a script	2-22
Running a TSP file	2-23
Retrieving scripts from the Series 2600	2-23
Instrument Console	2-24
File management tasks	2-30

Using the expanded system **2-33**
Source-measure voltage and current 2-33
Read and write to Digital I/O port 2-33
Display user-defined messages 2-34

User scripts **2-34**
Script examples 2-34
Creating a user script 2-36
Saving a user script 2-37
Running a user script 2-38
Modifying a user script 2-40
Script management 2-40

Factory scripts **2-42**
Running a factory script 2-42
Modifying a factory script 2-42
Differences: Remote versus local state 2-43
Memory considerations for the runtime environment 2-44

Test Script Language (TSL) reference **2-45**
Reserved words 2-45
Variables and types 2-45
Operators 2-46
Tables/arrays 2-47
Functions 2-46
Precedence 2-48
Logical operators 2-48
Concatenation 2-49
Branching 2-50
Loop control 2-51
Standard libraries 2-53

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 Instruments' 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). 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 TSL 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 TSL. The command set for a TSP-enabled instrument is referred to as 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 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 run-time 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 advanced 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 messages 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 name 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" later in this section 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` represents 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 they are 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 scripts 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)   --The .. operator concatenates two strings.
        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 the topic below, "What is a script?"

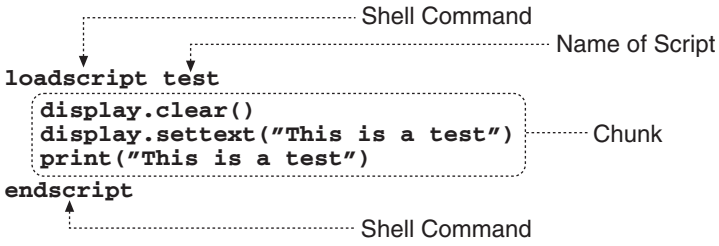
What is a script?

The Series 2600 utilizes a Test Script Processor (TSP) to process and run individual chunks or 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 Series 2600 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()`

Figure 2-1
Script example



A script is loaded into the Series 2600 System SourceMeter where it can be run. Running a script at the SourceMeter is faster than running a test program from the PC. The piecemeal 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" later in this section.

There are two types of scripts: Factory scripts and user scripts. A factory script was created by Keithley Instruments at the factory and stored in non-volatile memory of the Series System 2600 SourceMeter. Factory scripts cannot be removed from non-volatile memory. 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" later in this section). User scripts are loaded into the Series 2600 System SourceMeter run-time environment where they can be run and/or saved to 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 Series 2600 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 anonymous script created and loaded by the user is also placed in the run-time environment. Keep in mind that only one anonymous script, referred to as the active-script, can be in the run-time environment. If another anonymous script is created and loaded, it will replace the old anonymous script in the run-time environment

Non-volatile memory

After a new or modified user script is loaded into the Series 2600, 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 Series 2600 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 Series 2600. 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 Series 2600 System SourceMeters in the test system. There are three levels of programming (described later in this section):

- – Non-scripted chunks are executed one line at a time by the PC.
- User scripts – A program script is created and loaded into the Series 2600 System SourceMeter, where it is then run.
- Interactive script – 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 Instruments at the factory) are permanently stored in non-volatile memory of the Series 2600. User-created scripts can also be stored in non-volatile memory.

When the Series 2600 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 user's own program. Test data (e.g., a reading) is returned from the Series 2600 to the PC.

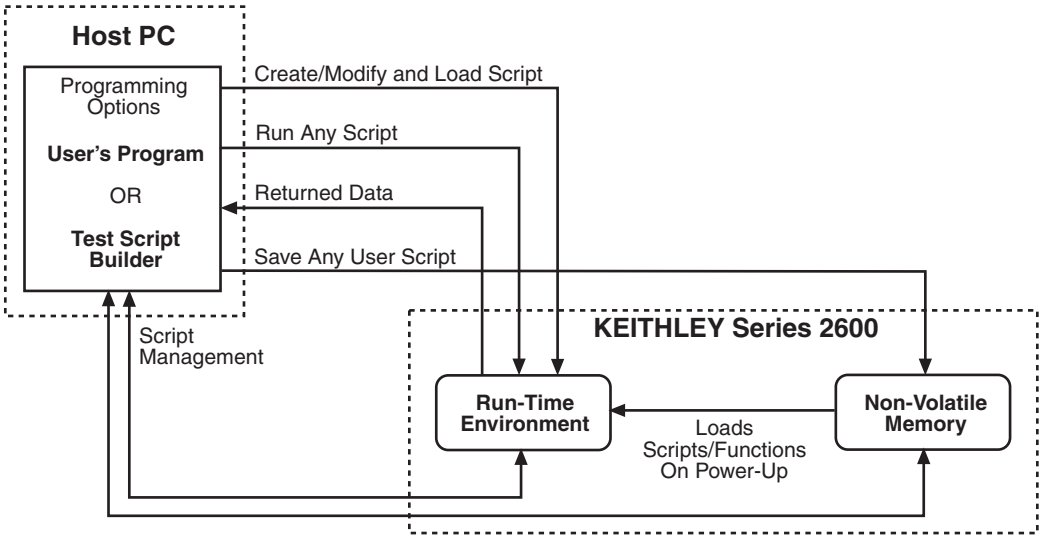
NOTE It is common practice to say that a script is run. In actuality, it is the chunk in the script that is being run (executed).

A user script can be created using the Test Script Builder or the user's 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 Series 2600. If it is not saved, the Series 2600 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 Test Script Builder software

To install the TSB software, close all programs, place the CD (Keithley Instruments part number: KTS-850) into your CD-ROM drive and follow the on-screen instructions. If your web browser does not start automatically and display a screen with software installation links, open the index.html file found on the CD using your web browser.

System connections

Up to 16 Series 2600 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 Series 2600 instruments. For the RS-232, a straight-through RS-232 cable terminated with DB-9 connectors is used to connect the PC to one of the Series 2600 instruments.

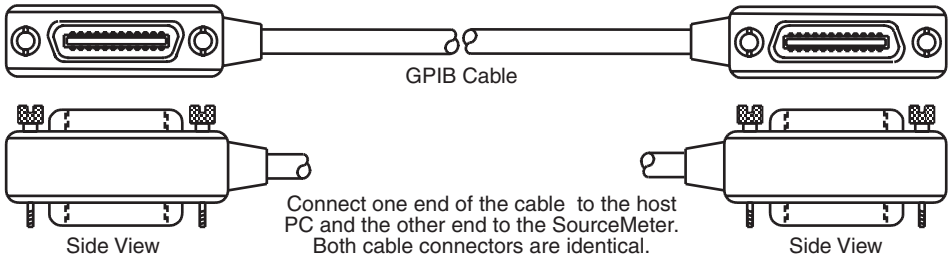
Keep in mind that the GPIB or RS-232 cable is only required to be connected to one of the Series 2600 instruments. Communication to the other Series 2600 instruments can be accomplished via the TSP-Link (see Section 9).

GPIB

GPIB connections

To connect the Series 2600 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 Instruments are the Model 7006 and Model 7007.

The GPIB cable connectors are stackable. For additional non-Series 2600 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.

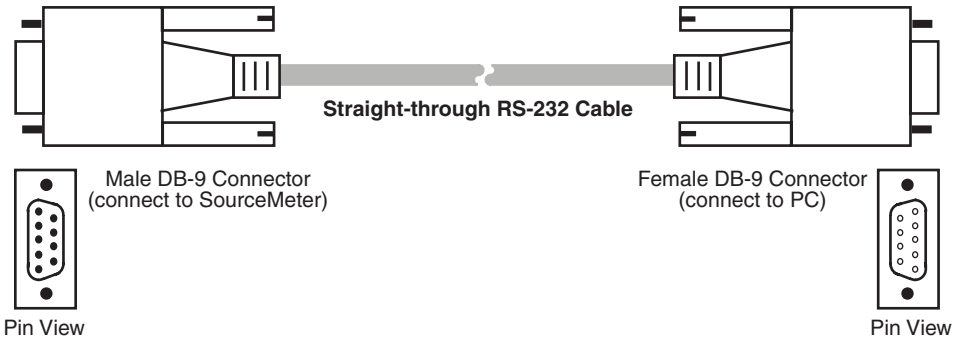
The GPIB address is saved in non-volatile memory. The address value will not change when power is cycled or a reset command (`reset` or *RST) is sent.

RS-232

RS-232 connections

To connect the Series 2600 to the RS-232 interface, use a straight-through RS-232 cable (see Figure 2-4). Figure 2-3 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.

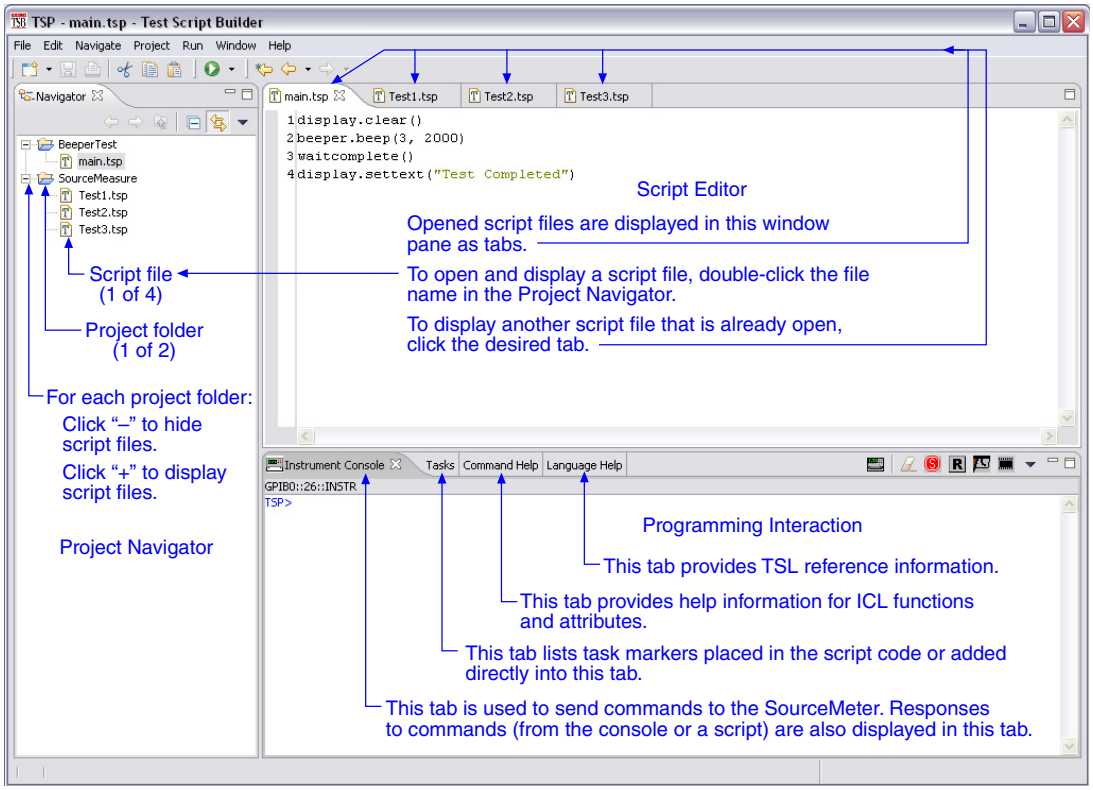
Script Editor

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” later in Section 2 for details on using the other tabs.

Figure 2-5
Test Script Builder (example)



Starting Test Script Builder

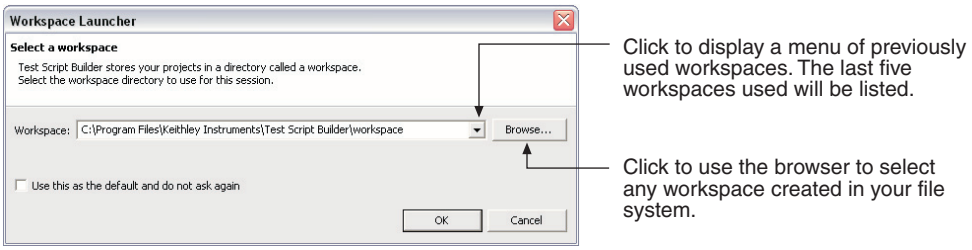
Make sure the SourceMeter is properly connected to the PC (see "System connections," described earlier in this section) and it is turned on. On the PC desktop, double-click the Test Script Builder icon to begin:

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” later in Section 2 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 toolbar on the Instrument Console tab is used to open or close communications.

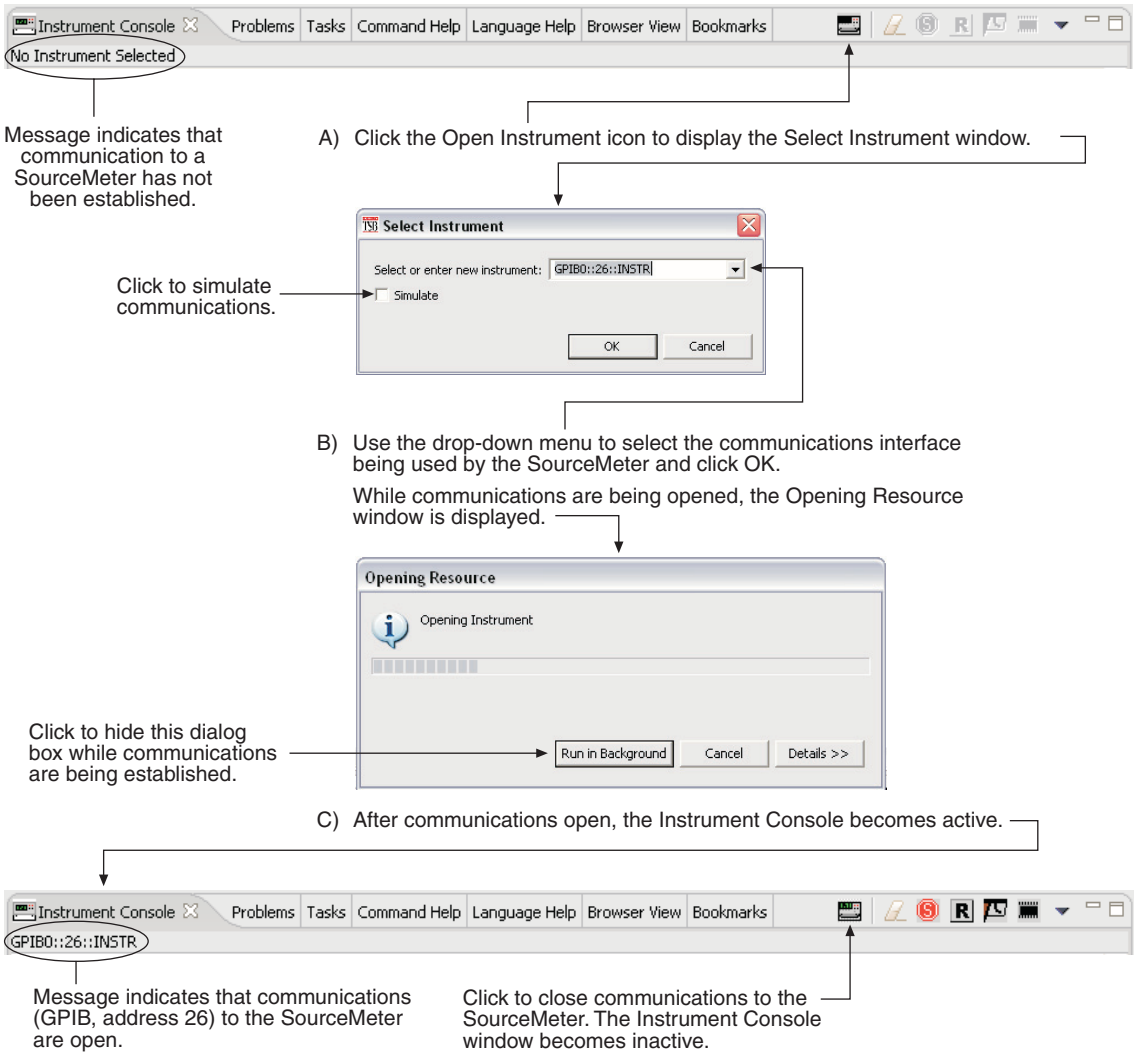
Figure 2-6 illustrates how to open and close communications. The following details supplement the information in the drawing:

The Select Instrument window has a drop-down menu to select the GPIB or RS-232 interface being used by the Series 2600.

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” later in this section for details on using the Menu icon.

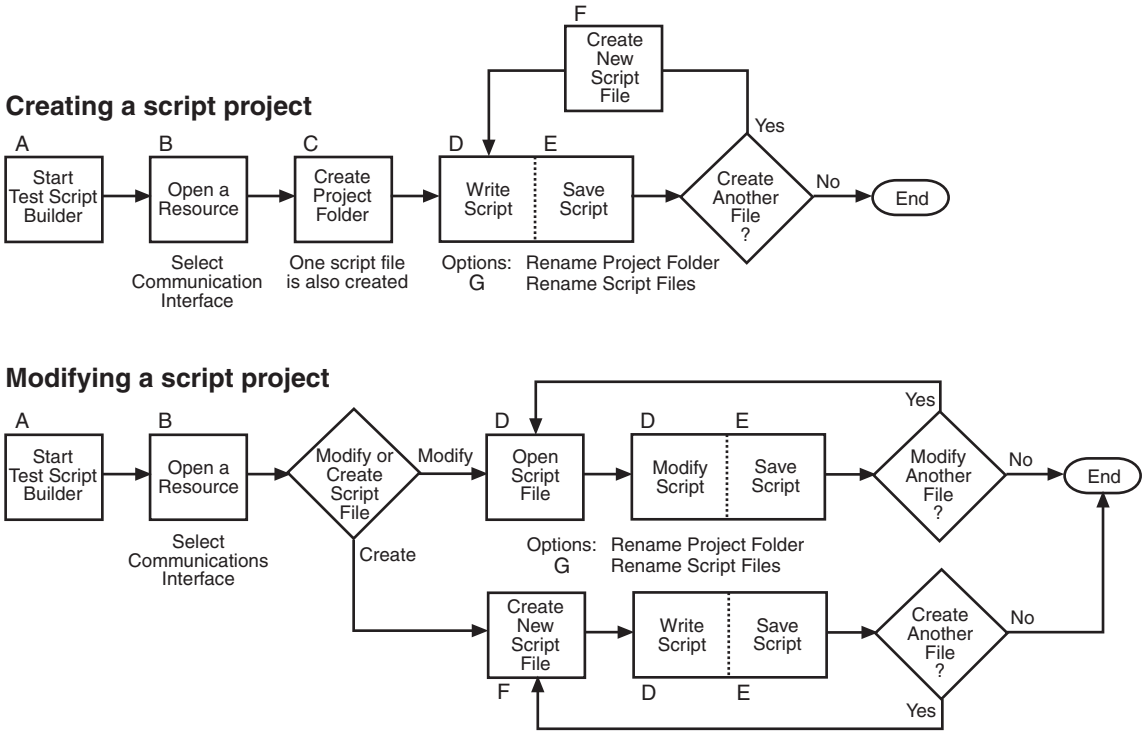
Figure 2-6
Opening and closing communications



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"

B "Opening communications"

C "Creating a project folder"

D "Writing or modifying a script"

E "Saving a script"

F "Creating new script files"

G "Renaming a project folder and/or script file"

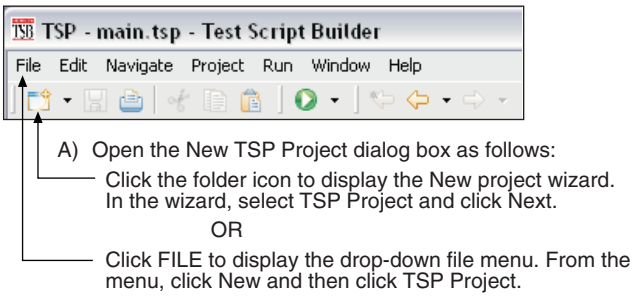
Creating a project folder

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

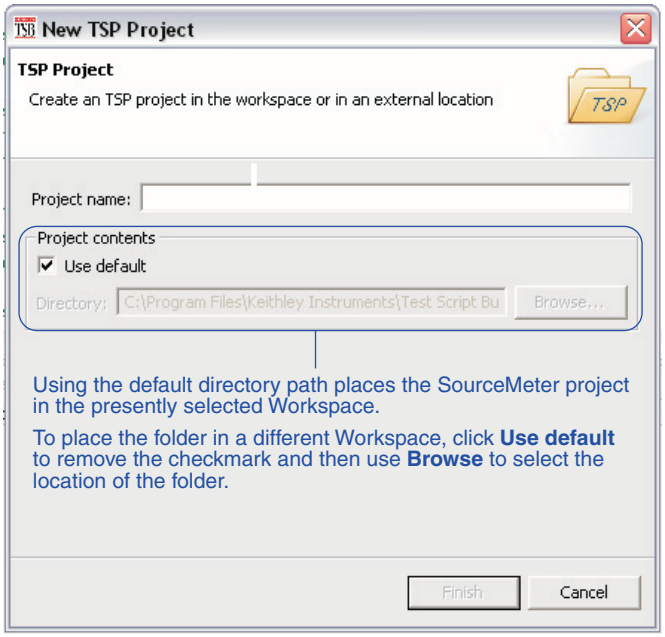
- The project folder is added to the Project Navigator.
- A script file (named "main") is created and placed in the project folder.
- The script file (which has no code) is opened and displayed in the Script Development area of the Test Script Builder.

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



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 Series 2600. See details on “User scripts” later in this section.

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

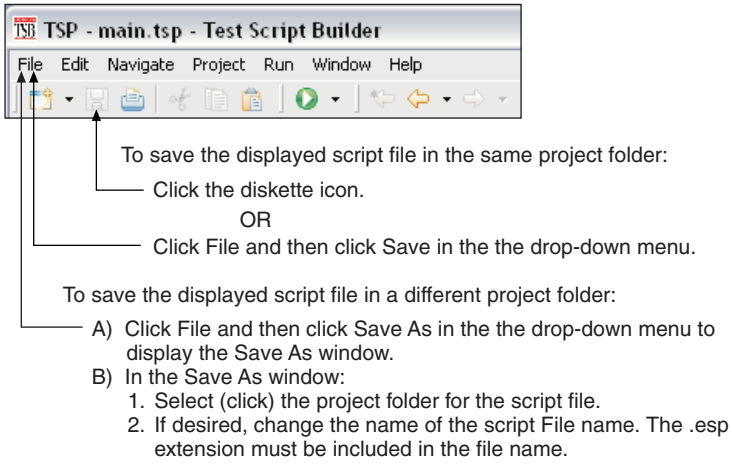
To modify an existing script file, it must be open. Open script files are presented as tabs in the Script Editor. To open and display a script file, click the file name in the Project Navigator. To display a different script file that is already open, click the appropriate tab at the top of the Script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

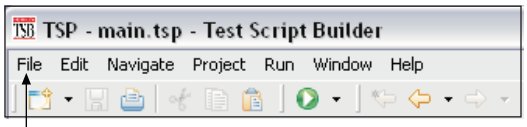
Figure 2-9
Saving a script in the Test Script Builder



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



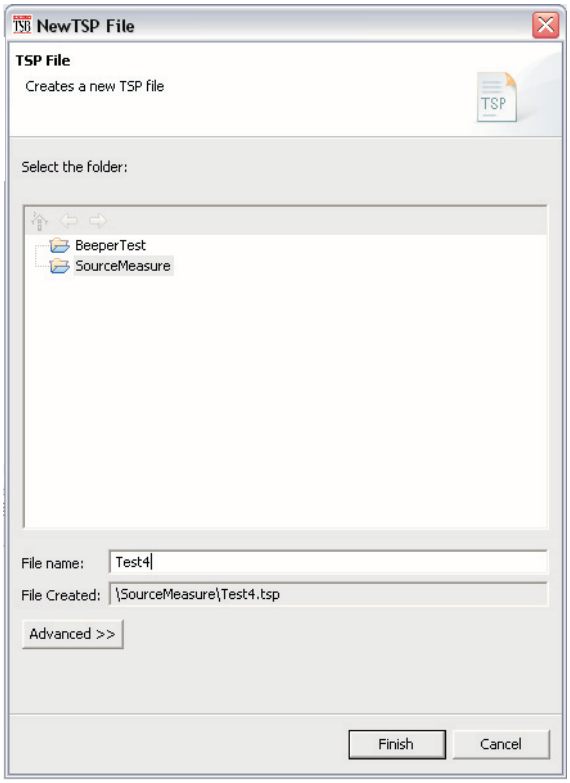
A) Open the New TSP File window as follows:
Click FILE to display the drop-down file menu. From the menu, click New and then click TSP File.

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.

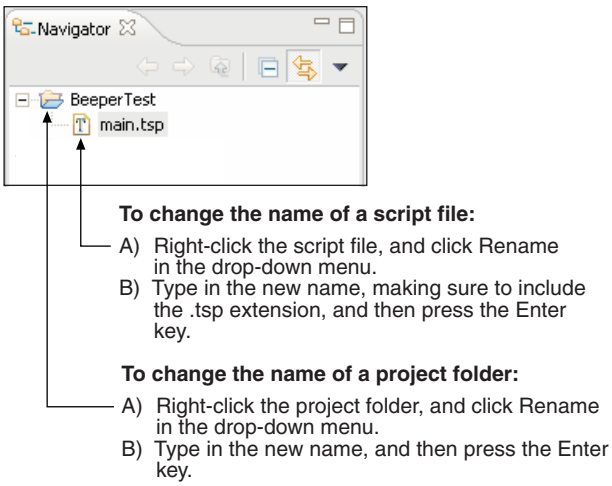
C) Type in a file name (e.g., Test4) and click Finish:



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



Script launch configuration

A script is to be loaded into the Series 2600 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 Series 2600: 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.

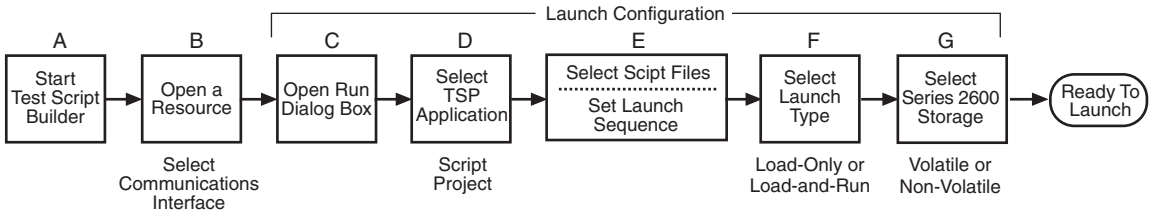
When a script project is created, the launch is configured initially as follows:

- 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 Series 2600. The script will be lost when the Series 2600 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” later in this section.

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"

B "Opening communications"

C "Displaying the launch configuration window"

D "Selecting a configuration"

E "Selecting script files and launch order"

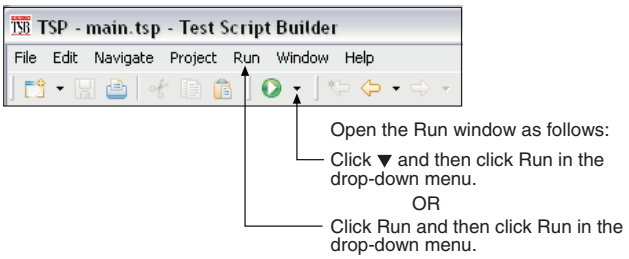
F "Selecting the type of launch"

G "Storing the script"

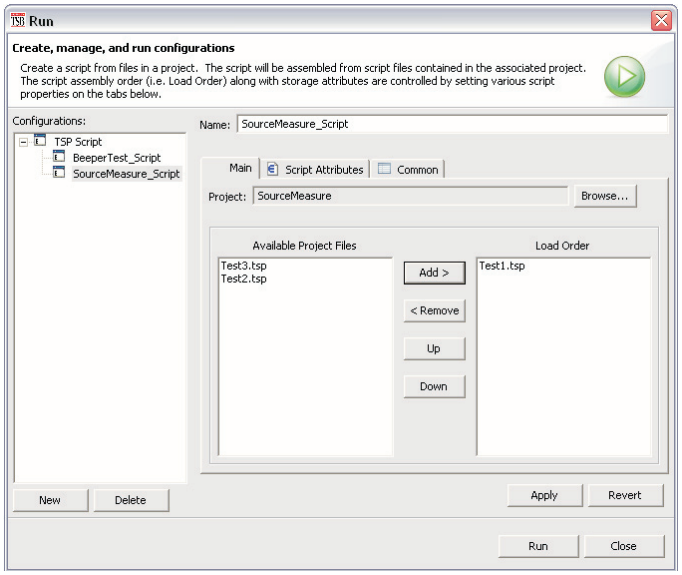
Displaying the launch configuration window

A launch is configured from the **Run** dialog box. As shown in Figure 2-13, use the toolbar 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 "SourceMeasure_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 Series 2600, 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 Series 2600 is powered on.

Storing the script

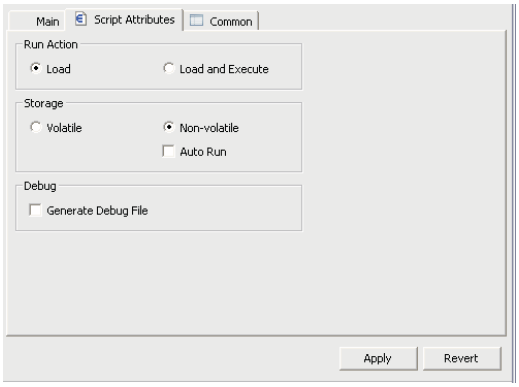
When a script is launched it can be stored in the volatile or non-volatile memory of the Series 2600. If stored in volatile memory, it will be lost when the SourceMeter power is turned off. If stored in non-volatile memory, it will not be lost when the power is turned off.

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)

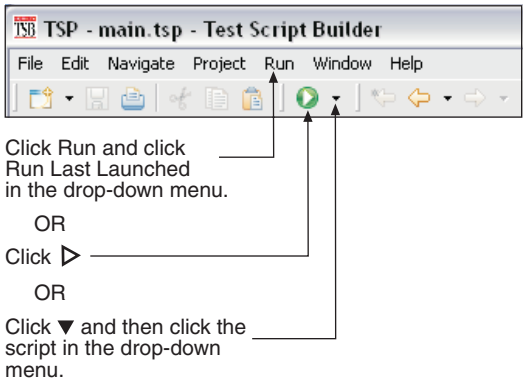


Launching a script

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 relaunched directly from the toolbar located at the top of the Test Script Builder. Figure 2-15 explains how to relaunch a script from the toolbar.

Figure 2-15
Relaunching a script from the Test Script Builder toolbar



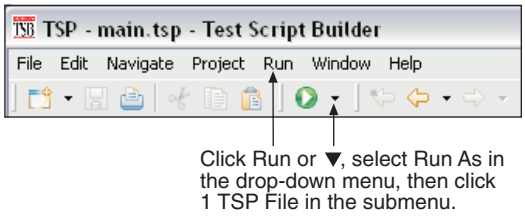
Running a TSP file

A TSP (.tsp) file does not have to be launched (loaded) into the Series 2600 in order to be run. The code for a TSP file can simply be sent to the Series 2600 and executed. The TSP file will not reside in the Series 2600 (it is not saved in volatile or non-volatile memory). A TSP file can be run from the Project Navigator or from the toolbar 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 toolbar as explained in Figure 2-16.

Figure 2-16
Re-launching a script from the Test Script Builder toolbar



A TSP file can also be run from the Menu icon on the Instrument Console toolbar. For details, see "Instrument Console icons" later in this section.

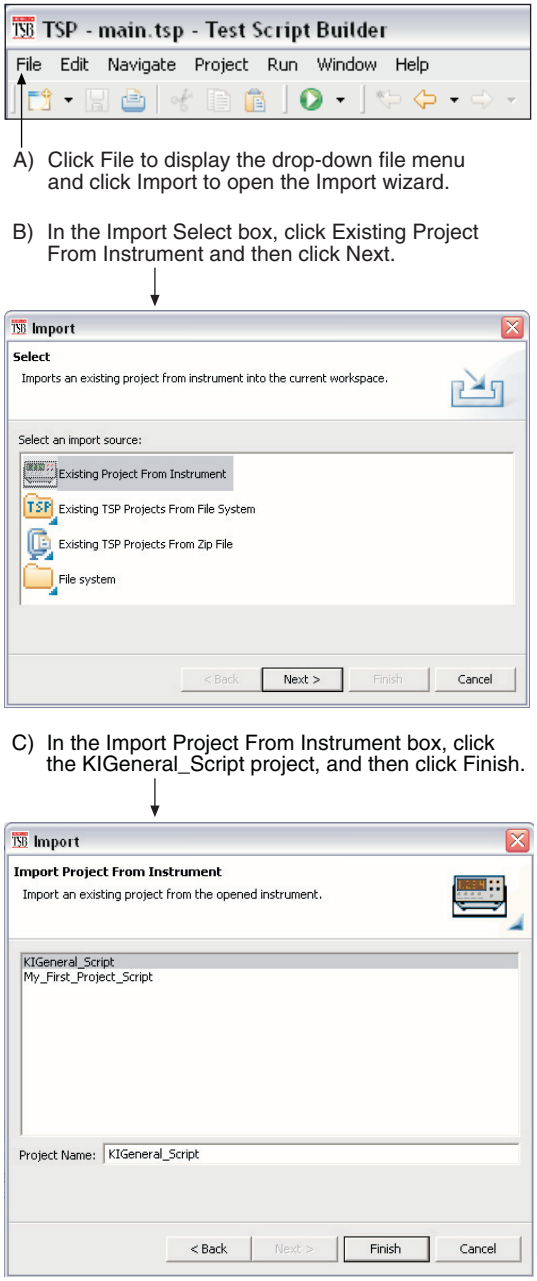
Retrieving scripts from the Series 2600

A user script or factory script can be retrieved from memory of the Series 2600. 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 Series 2600. 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 Series 2600 as a user script using the same name or a new name. An imported factory script can only be loaded back into the Series 2600 as a user script.

Figure 2-17
Importing a script from memory of the Series 2600



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.

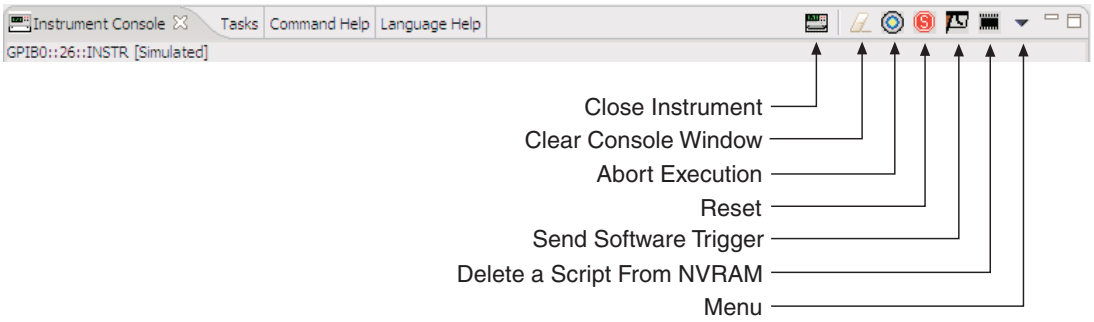
NOTE See "" for example code that can be sent from the Instrument Console.

Code and messages in the Instrument Console can be cleared by clicking the Clear Console Window icon. 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 will be active.

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 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 Instruments Flash Programmer is used to download firmware upgrades into the Series 2600. See "Flash programmer" later in this section 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" later on in this section). 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 2-18 and Figure 2-19 show partial screen shots 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 provides details on ICL functions and attributes (see Section 4 of this 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 provides details on the Test Script Language (TSL); see "Test Script Language (TSL) reference" later in 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 Instruments website (www.keithley.com).

Bookmarks tab

Tasks tab

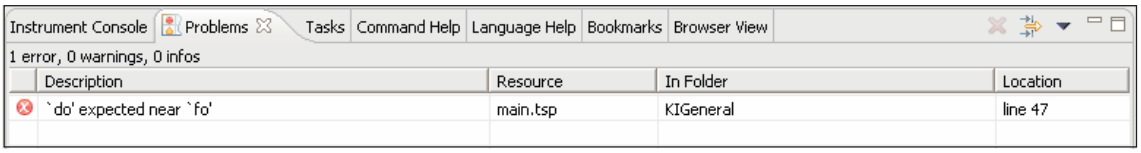
This tab displays bookmarks that are placed in the Script Editor by the user. A bookmark is placed for a line of code in the left-hand margin of the Script Editor. Right-click the line number for the code and select Add Bookmark from the mouse menu. In the Add Bookmark window, type in a bookmark name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

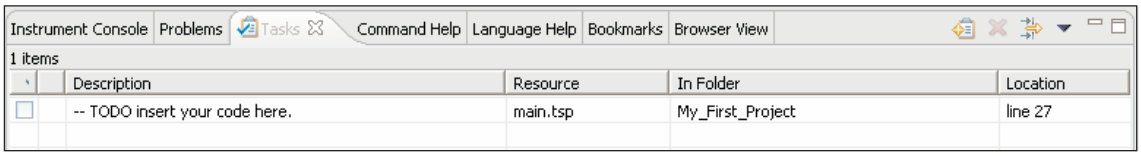
The Bookmarks tab in Figure 2-19 shows an example of using bookmarks. Each bookmark in the tab is linked to a function for a script file that exists in the Project Navigator. When a bookmark is clicked, the first line for that function will be displayed and highlighted in the Script Editor.

Figure 2-18
Programming interaction tabs: Problems, Tasks and Command Help

Problems tab:



Tasks tab:



Command Help tab:

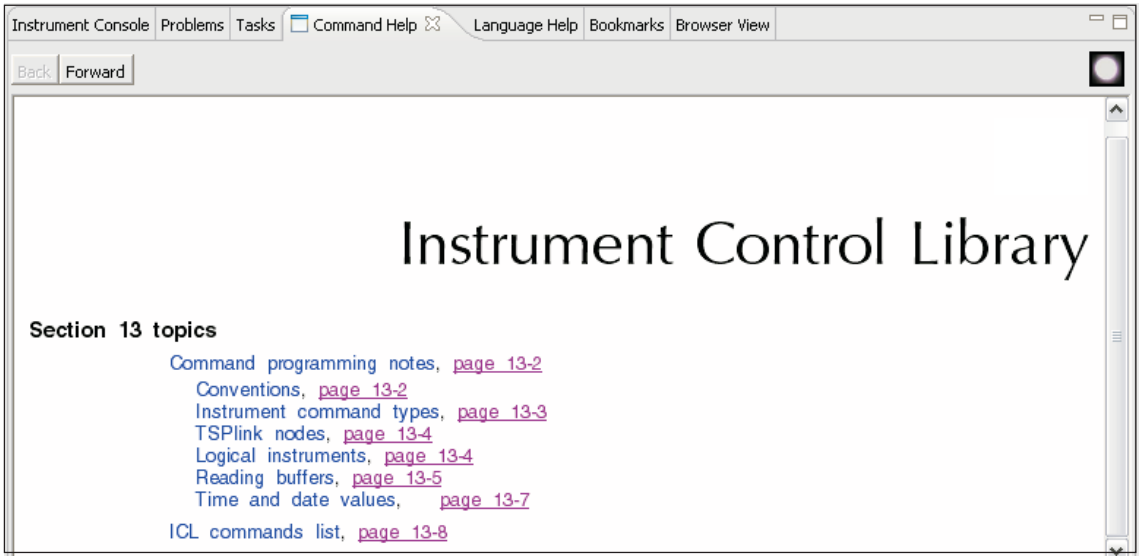
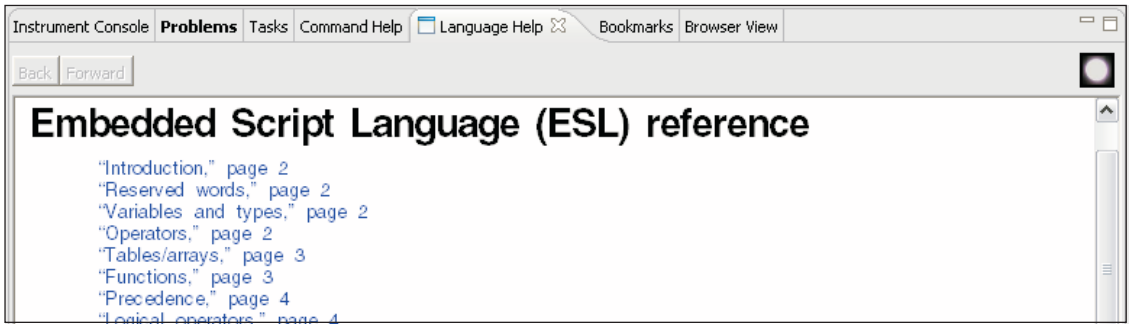
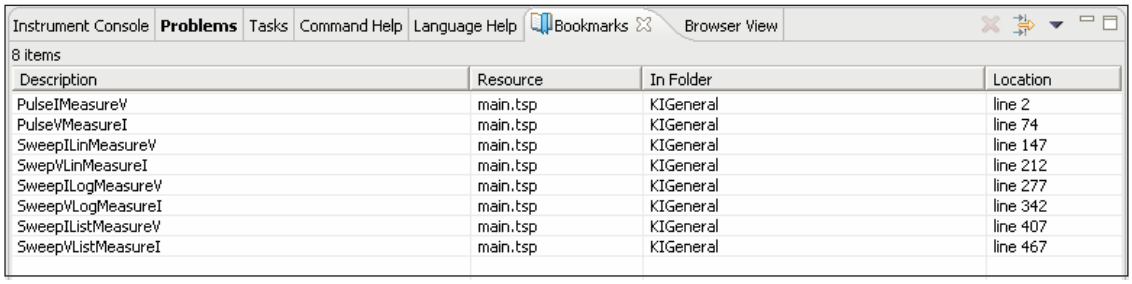


Figure 2-19
Programming interaction tabs: Language Help, Bookmarks, Browser View

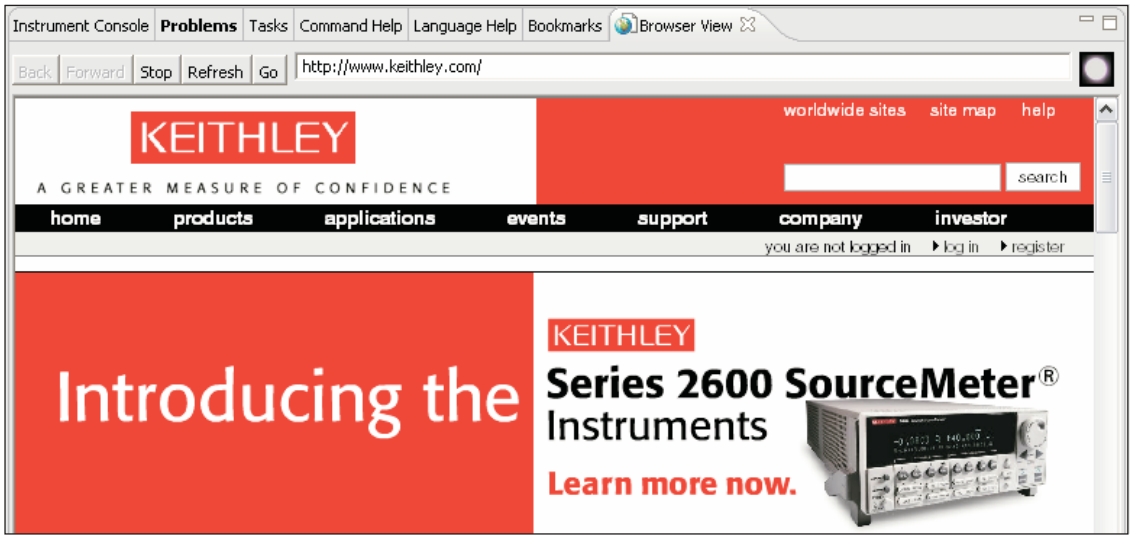
Language Help tab:



Bookmarks tab:



Browser View tab:



Flash programmer

When a firmware upgrade for the Series 2600 becomes available, it can be downloaded from the Keithley Instruments website (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 Series 2600 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 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" in Section 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" earlier in this section for details on file management tasks). A script project can also be imported from a Series 2600 into Test Script Builder, where it can be modified (for details, see "Retrieving scripts from the Series 2600," previously described in this section).

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

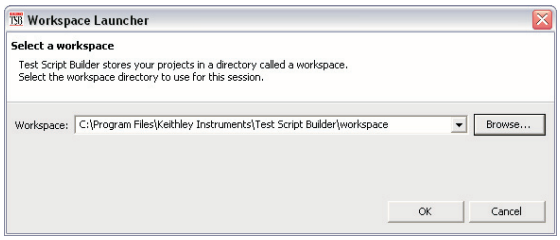
Additional workspaces can be created anywhere in your file system. A new workspace is simply a new folder for project files. A new folder for a workspace can be made from TSB as follows:

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 2-20A).
2. Click the Browse button to open the Select Workspace Directory browser and select the location for the new folder. Figure 2-20B 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 reopen using the new workspace.

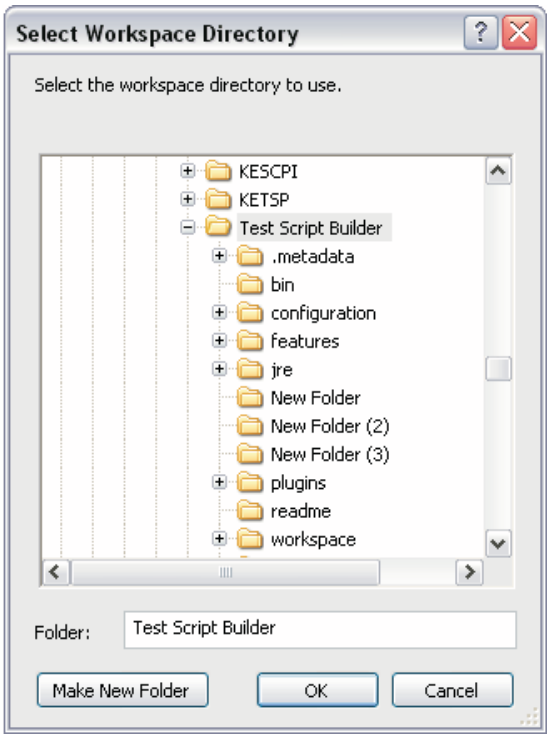
There will not be any projects residing in the Project Navigator for the new workspace. New projects and script files can be created as explained in "Creating and modifying a script" earlier in this section. A project (along with its script files) can be imported into the new workspace from another workspace folder. See "Importing a project from another workspace" that follows.

Figure 2-20
Workspace Launcher and Select Workspace Directory

A)



B)

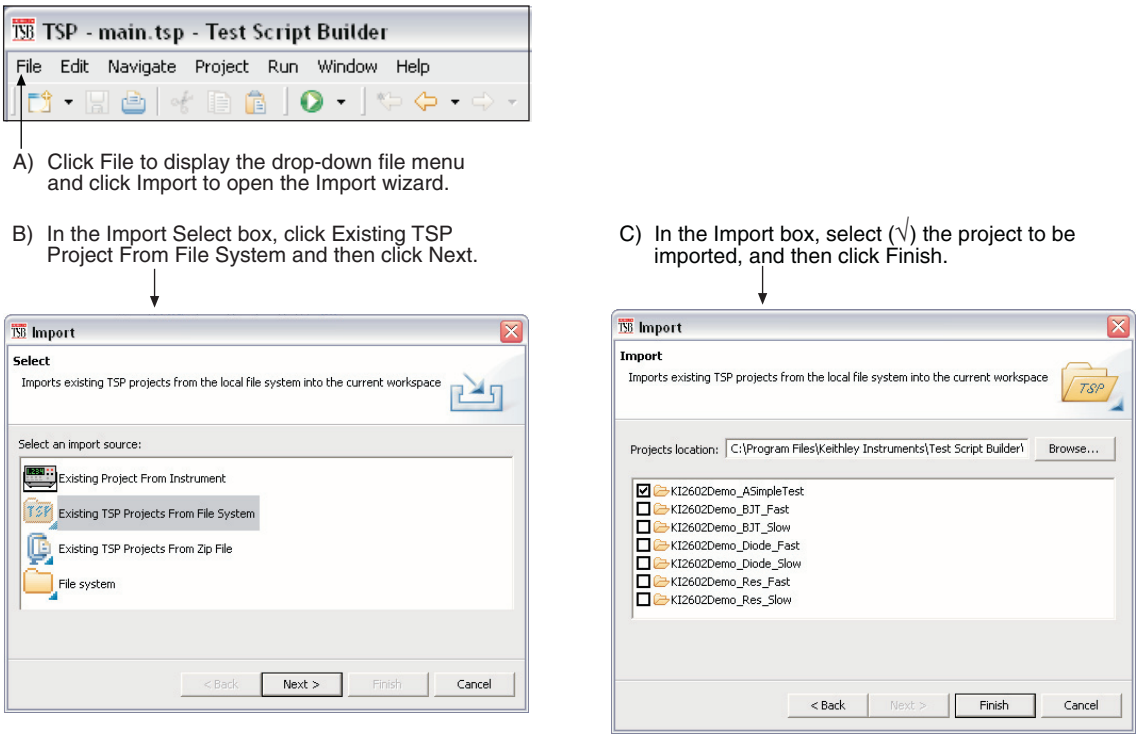


Importing a project from another workspace

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 2-21, which imports a project named KI2602Demo_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 2-21
Importing a project from another workspace folder



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 2-20A).
2. Click the Browse button to open the Select Workspace Directory browser (Figure 2-20B) and select the workspace folder. TSB will shut down and then reopen using the selected workspace.

Deleting projects and/or script files

Deleting a project

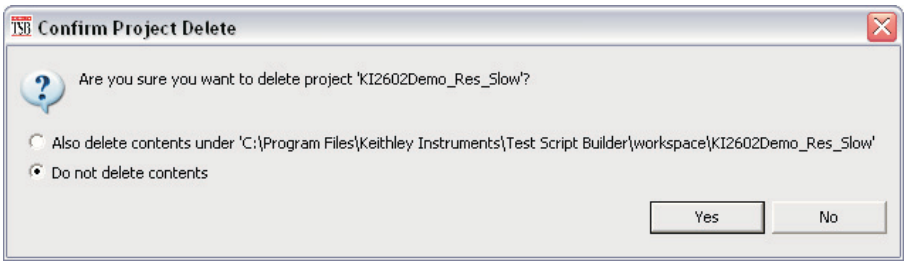
To delete a project, right-click the project in the Project Navigator and then click Delete in the mouse menu to display the Confirm Project Delete window (see Figure 2-22).

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" described earlier in this section).

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

Figure 2-22
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.

Using the expanded system

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" (later in this section) 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 Series 2600 and can be saved in non-volatile memory. Scripts not saved in non-volatile memory will be lost when the Series 2600 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

`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`
	`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`

NOTE When creating a script using the Test Script Builder, only the chunk is typed in as shown above. See "Using Test Script Builder" earlier in this section 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 Series 2600 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

`smua.measure.nplc = 0.5`
`function sourcev(v)`
`  smua.source.levelv = v`
`  i = smua.measure.i()`
`  print(i)`
`  return(i)`
`end`
	`loadscript`
	`  smua.measure.nplc = 0.5`
	`  function sourcev(v)`
	`  smua.source.levelv = v`
	`  i = smua.measure.i()`
	`  print(i)`
	`  return(i)`
	`  end`
	`endscript`

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

Script Chunk Fragment (Test Script Builder or User's Program)

```
--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
```

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 `loadandrunscript`, and `endscript`.

Load only – The following scripts will load only into the run-time environment of the Series 2600. The script on the left is anonymous, while the one on the right is named (where `name` is the user-defined name):

```
loadscript
    (chunk)
endscript
```

```
loadscript name
    (chunk)
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 anonymous, while the script on the right is named (where `name` is the user-defined name):

```
loadandrunscript
    (chunk)
endscript
```

```
loadandrunscript name
    (chunk)
endscript
```

Details on `loadscript` and `loadandrunscript` 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 Series 2600 is turned off. After loading the anonymous 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 Series 2600 is powered on and all autorun scripts have been executed. For details, see "Autoexec script" and "Autorun scripts" later in this section.

```
loadandrunscript
loadsandrunscript 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" later in this section).

Saving a user script

A created and loaded script does not have to be saved in non-volatile memory of the Series 2600 before it can be run. However, an unsaved script will be lost when the Series 2600 is turned off.

Saving a named script

Only a named script can be saved in non-volatile memory of the Series 2600. 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 Series 2600 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 the anonymous script

There can only be one anonymous script in the run-time environment. If another anonymous script is created and loaded, the previous anonymous script will be removed from the run-time environment. Use one of the following commands to execute the chunk of the last loaded anonymous 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 Series 2600 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 Series 2600 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 Series 2600 is turned on, the “test5” script will automatically load and run.

NOTE The `loadandrunscript name` command sets the autorun attribute for that script to "yes." To cancel autorun, set the autorun attribute to "no" and save the script.

Autoexec script

One script can be designated as the autoexec script. When the Series 2600 is turned on, the autoexec script will start after all the autorun scripts have run.

```
loadscript autoexec
loadandrunscript autoexec
```

An autoexec script can be formed by creating a new script and naming it `autoexec` (as shown above using `loadscript` or `loadandrunscript`). After loading the new script, send the `autoexec.save()` command to save it in non-volatile memory. See "Creating a user script" (described earlier in this section) 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 Series 2600 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 Series 2600 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" (below) 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 Series 2600" described earlier in Section 2).

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 Series 2600 using the same name or a new name.

The following command returns a catalog listing of the user scripts stored in the Series 2600:

```
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 Instruments 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" (found earlier in this section) 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 "KIGeneral"

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

Running a factory script from the Series 2600 front panel controls

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

NOTE An imported factory script can only be loaded back into the Series 2600 as a user script.

The following command returns a catalog listing of the factory scripts stored in the Series 2600:

```
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 “KIGeneral”:

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

Differences: Remote versus local state

The Series 2600 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 16 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 for the runtime environment

The Series 2600 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 x 15) + (2500 x 15) = 52,500 bytes or 52.5 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 website for the Lua Programming Language, for more information. Also, 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)	

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))
```

Here is 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
```

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]             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
```

Tables can be indexed using element names instead of numeric indices. Since functions are first-class variables, tables can be used to create "pseudo-classes." Classes are often used in object-oriented programming.

Below is a table used to create a circle pseudo-class. It has 3 elements:

clr: a string containing the color of the circle

diam: a number containing the diameter of the circle

setdiam: a function, or method, used to change the diameter

```
circle = {clr = "red", diam = 1, setdiam = function(d)
                        circle["diam"]=d end}
print(circle["clr"])     -- Index using a string; print the clr property.
print(circle["diam"])    -- Index using a string; print the diam property.
circle["setdiam"](2)     -- Change the diam element by calling setdiam method.
print(circle.diam)       -- circle["diam"] is the same as circle.diam; simpler syntax
circle.setdiam(3)        -- Change the diameter of the circle again.
print(circle.diam)       -- Print diam property again using simple syntax.
```

Output of code above:

```
red
1
2
3
```

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 and y <= z       (a < y) 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
```

For example, 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 are 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 0 evaluates
    print("Zero is true!")         false. In TSL, “nil” is false and everything else is true.
else
    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

`print(x)`	Prints the argument x to the active host interface, using the `tostring()` function to convert x to a string.
`collectgarbage([limit])`	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 **Note** for more information.
`gcinfo()`	Returns the number of Kbytes of dynamic memory that TSP is using.
`tonumber(x [,base])`	Returns x converted to a number. If x is already a number, or a convertible string, then the number is returned; otherwise, it returns nil. An optional argument specifies the base to interpret the numeral. The base may be any integer between 2 and 36, inclusive. In bases above 10, the letter 'A' (in either upper or lower case) represents 10, 'B' represents 11, and so forth, with 'Z' representing 35. In base 10, the default, the number may have a decimal part, as well as an optional exponent. In other bases, only unsigned integers are accepted.
`tostring(x)`	Receives an argument of any type and converts it to a string in a reasonable format.
`type(v)`	Returns the type of its only argument, coded as a string. The possible results of this function are: nil, number, Boolean, table, or function.

NOTE: 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.

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.

`string.byte(s [,i])`	Returns the internal numerical code of the i-th character of string s, or nil if the index is out of range.
`string.char(i1, i1, …)`	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.
`string.format(fs, e1, e2, …)`	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 print family of ANSI C functions. The only differences are that the options/modifiers *, l, L, n, p, and h are not supported. The options c, d, E, e, f, g, G, I, o, u, X, and x all expect a numeric argument, where s expects a string argument. String values to be formatted with %s cannot contain embedded zeros.
`string.len(s)`	Returns the length of the strings.
`string.lower(s)`	Returns a copy of the string s with all uppercase letters changed to lowercase.
`string.rep(s, n)`	Returns a string that is the concatenation of n copies of the string s.
`string.sub(s, i [,j])`	Returns the substring of s that starts at i and continues until j. i and j may be negative. If j is absent, then it is assumed to be equal to –1, which is the same as the string length. In particular, the call `string.sub(s,1,j)` returns a prefix s with length j, and `string.sub(s, -i)` returns a suffix s with length i.
`string.upper(s)`	Returns a copy of the string s 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.

`math.abs(x)`	Returns the absolute value of the argument x.
`math.acos(x)`	Returns the principal value of the trigonometric arc cosine function of x.
`math.asin(x)`	Returns the principal value of the trigonometric arc sine function of x.
`math.atan(x)`	Returns the principal value of the trigonometric arc tangent function of x.
`math.atan2(y,x)`	Returns the principal value of the trigonometric arc tangent function of y/x.
`math.ceil(x)`	Returns the smallest floating-point number not less than x whose value is an exact mathematical integer.
`math.cos(x)`	Returns the trigonometric cosine function of x.
`math.deg(x)`	Returns the value of x in degrees, where x is in radians.
`math.exp(x)`	Returns the exponential function of x; that is, e^x, where e is the base of the natural logarithms.

`math.floor(x)`	Returns the largest floating-point number not greater than x whose value is an exact mathematical integer.
`math.log(x)`	Returns the natural logarithm function of x.
`math.log10(x)`	Returns the base-10 logarithm function of x.
`math.max(x, y, …)`	Returns the maximum value of its numeric argument(s).
`math.min(x, y, …)`	Returns the minimum value of its argument(s).
`math.mod(x, y)`	Returns an approximation to the mathematical value f such that f has the same sign as x, the absolute value of f is less than the absolute value of y, and there exists an integer k such that k*y+f = x.
`math.pi`	Variable containing the value of π (3.141592654).
`math.pow(x, y)`	Returns x^y.
`math.rad(x)`	Returns the value of x in radians, where x is in degrees.
`math.sin(x)`	Returns the trigonometric sine function of x.
`math.sqrt(x)`	Returns the non-negative square root of x.
`math.tan(x)`	Returns the trigonometric tangent function of x.
`math.frexp()`	Splits x into a fraction f and exponent n, such that f is 0.0 or 0.5 <= | f | <= 1.0, and f * 2^n is equal to x. Both f and n are returned; `f,n = math.frexp(x).`
`math.ldexp(x, n)`	Returns the inverse of the math.frexp() function; it computes the value x * 2^n
`math.random([x],[y])`	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].
`math.randomseed(x)`	Sets a “seed” for the pseudo-random generator. Equal seeds produce equal sequences of numbers.

This page left blank intentionally.

Section 3

DUT Test Connections

In this section:

Topic	Page
Input/output connectors	**3-2**
Input/output LO and chassis ground	**3-3**
Sensing methods	**3-5**
2-wire local sensing	3-6
4-wire remote sensing	3-7
Sense mode selection	3-8
Contact check connections	**3-8**
Multiple SMU connections	**3-10**
Guarding and shielding	**3-12**
Guarding	3-12
Noise shield	3-13
Safety shield	3-16
Using shielding and guarding together	3-18
Test fixture	**3-19**
Floating a SMU	**3-20**
Output-off states	**3-23**

Input/output connectors

The Keithley Instruments Series 2600 System SourceMeter® Models 2601, 2602, 2611, and 2612 use screw connectors for input and output connections to DUTs (devices under test). The Model 2602/2612 uses two connectors as shown in Figure 3-1 (one for each source-measure unit (SMU) channel). The Model 2601/2611 has only one connector for a single SMU. Models 2635 and 2636 use triax connectors as shown in Figure 3-2.

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

After making the wire connections from a connector to a DUT, reinstall the connector onto the rear panel 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 Series 2600 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 a SMU is 250V. Exceeding this level could damage the instrument and create a shock hazard. See "Floating a SMU" later in this section for details on floating the SMUs.

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

To prevent electric shock and/or damage to the SourceMeter, when connecting to a source with a greater current capability than the Series 2600, a fuse should be provided in-line with the SourceMeter input/output connectors rated no more than 3A.

Figure 3-1
2602/2612 input/output connectors

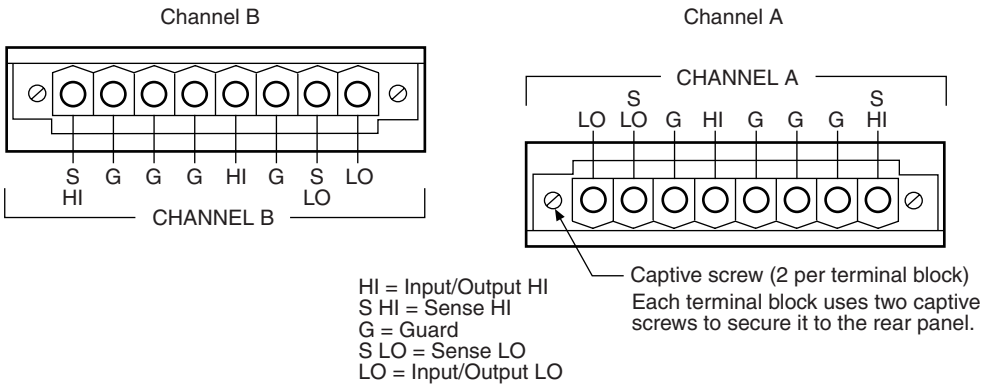
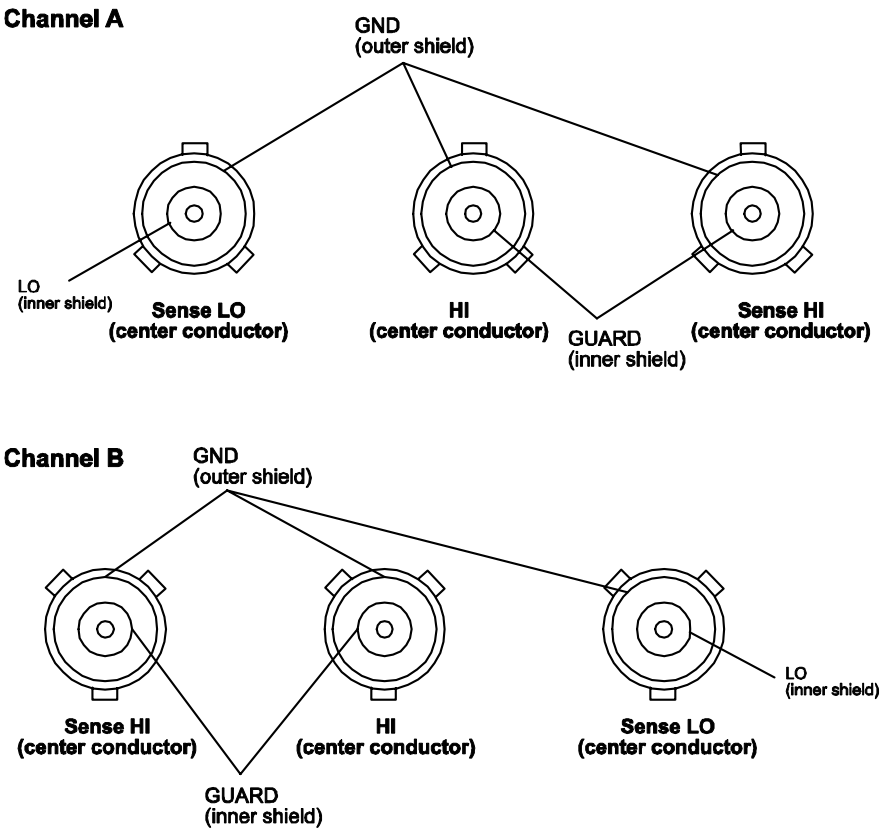


Figure 3-2
Model 2636 input/output connectors



Input/output LO and chassis ground

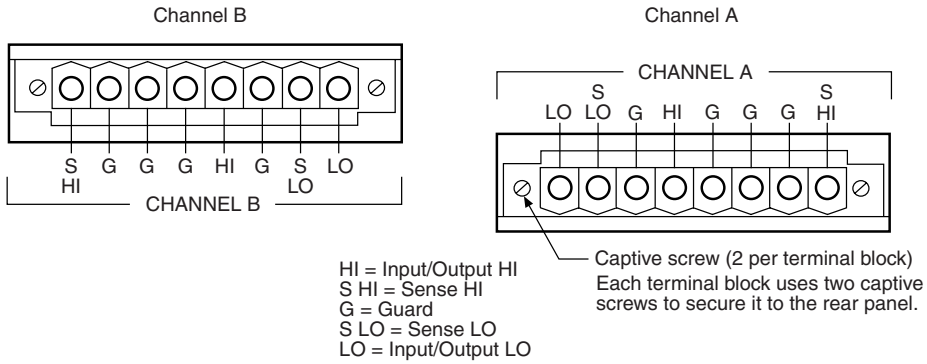
As shown in Figure 3-3, SMU input/output LOs are available at the rear panel terminal blocks. Input/output LOs are not connected between channels and are electrically isolated from chassis ground.

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 3-4) is used to isolate the SMUs from high frequencies that may be present on the chassis of the Series 2600. 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 Series 2600 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 Series 2600.
For Model 2636, connect to ground on the ground module not to the chassis screw.

Figure 3-3
Model 2602/2612 input/output LO and chassis ground terminals



Model 2636 input/output and chassis ground

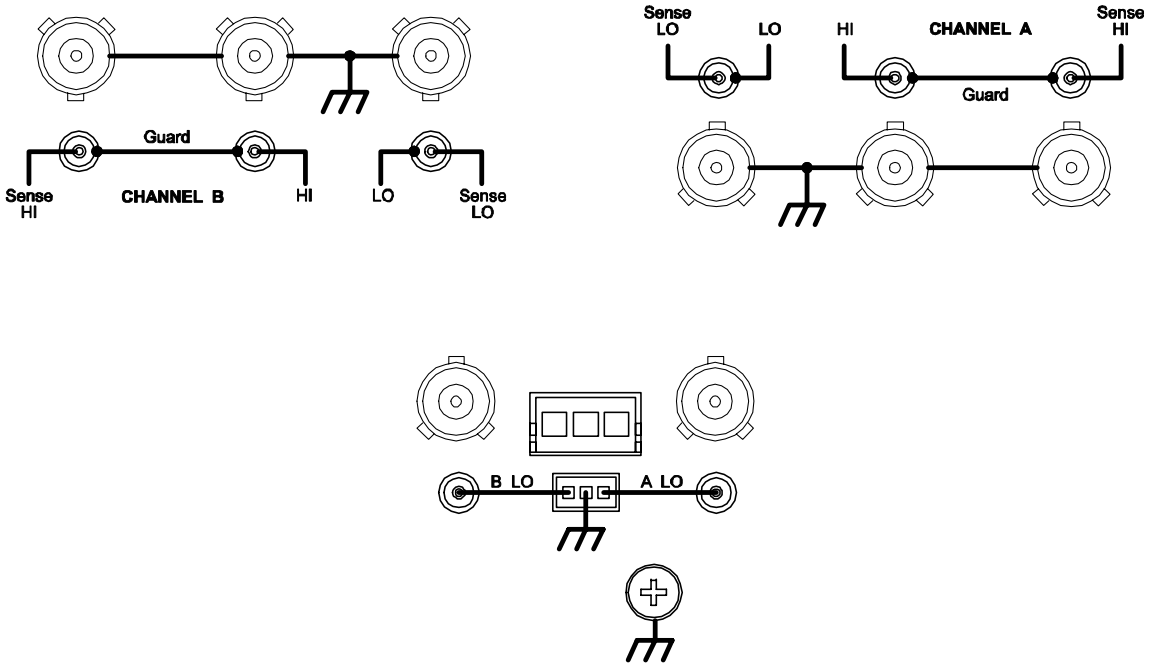
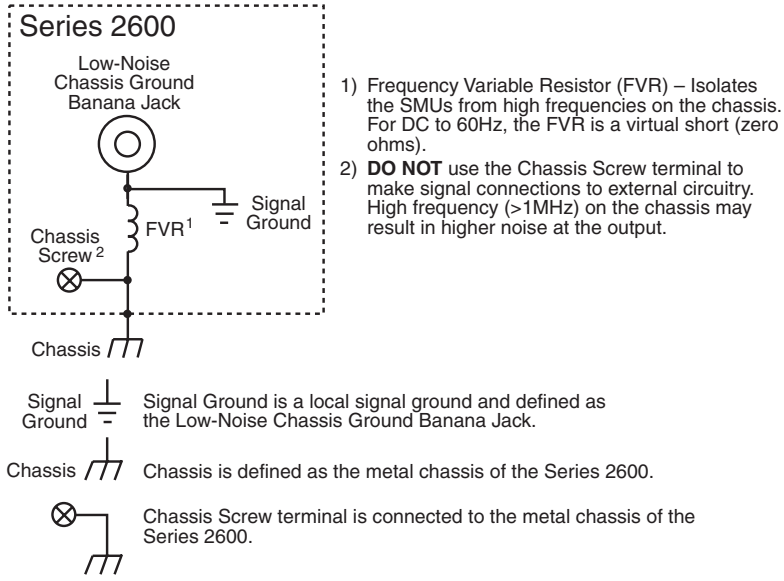
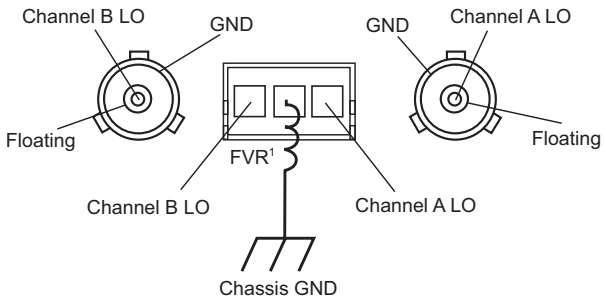


Figure 3-4
Model 2602/2612 Low-Noise Chassis Ground Banana Jack and Chassis Screw



Model 2636



WARNING ***When connecting to the model 2611, 2612, 2635 and 2636 SMU outputs, with cables not rated for voltages above 42V, such as the 2600-ALG-2, you must disable the high voltage output by using the INTERLOCK function as defined in section 10 of this manual. Leaving the high voltage enabled while not properly insulating the external connections to the unit poses a shock hazard which could cause serious injury to the user. It is also recommended that the LO connection terminal not be allowed to float by connecting it to signal ground or another known signal reference.***

Sensing methods

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

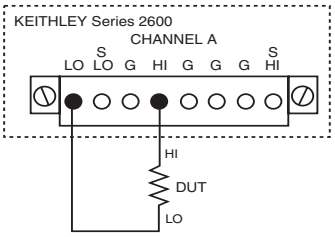
NOTE The default sense setting is 2-wire local. See "Sense mode selection" later in this section to check and or change the sense mode.

2-wire local sensing

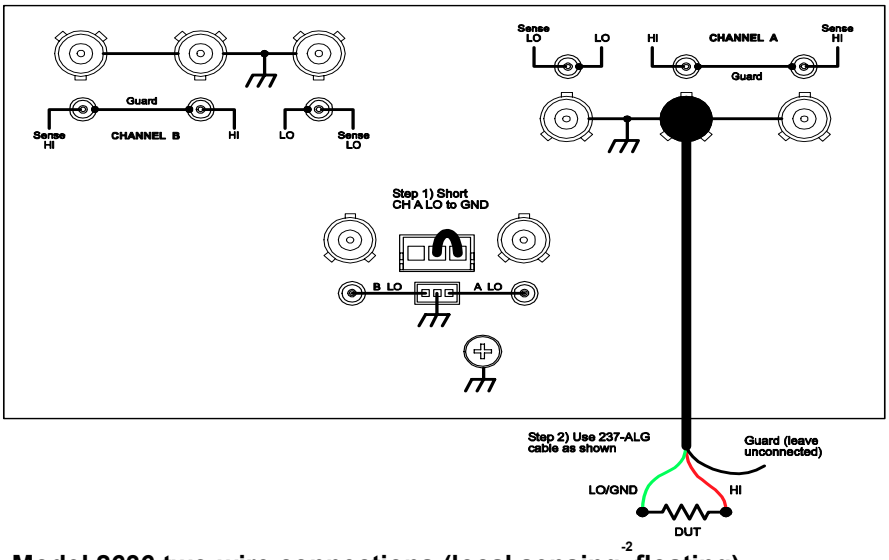
Two-wire local sensing (as shown in Figure 3-5) can be used for the following source-measure conditions:

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

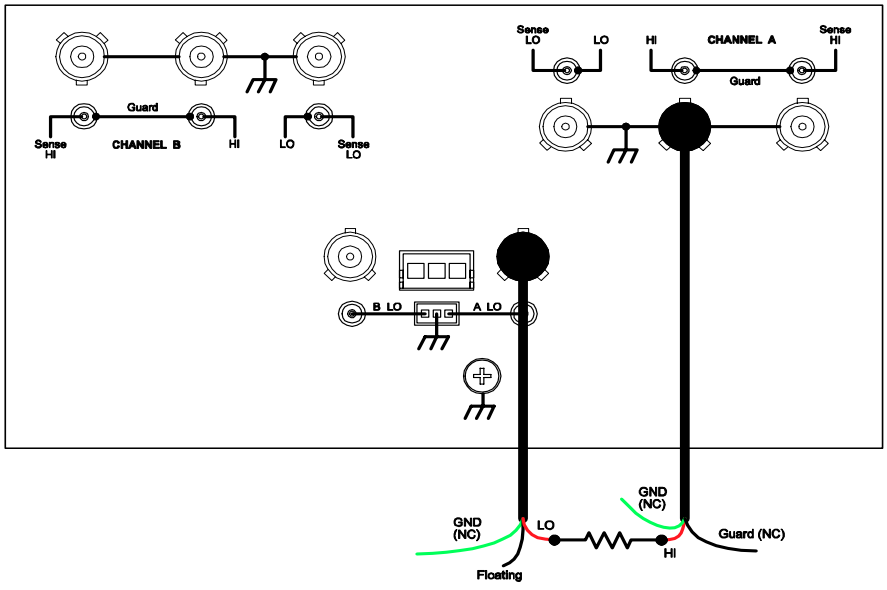
Figure 3-5
Model 2602/2612 two-wire connections (local sensing)



Model 2636 two-wire connections (local sensing, non-floating)



Model 2636 two-wire connections (local sensing, floating)



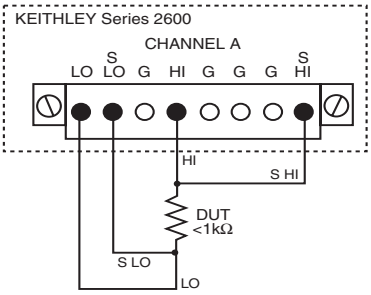
4-wire remote sensing

When sourcing and/or measuring voltage in a low-impedance test circuit (see Figure 3-6), there can be errors associated with IR drops in the test leads. Voltage source and measure accuracy are optimized by using 4-wire remote sense connections. When sourcing voltage, 4-wire remote sensing ensures that the programmed voltage is delivered to the DUT. When measuring voltage, only the voltage drop across the DUT is measured.

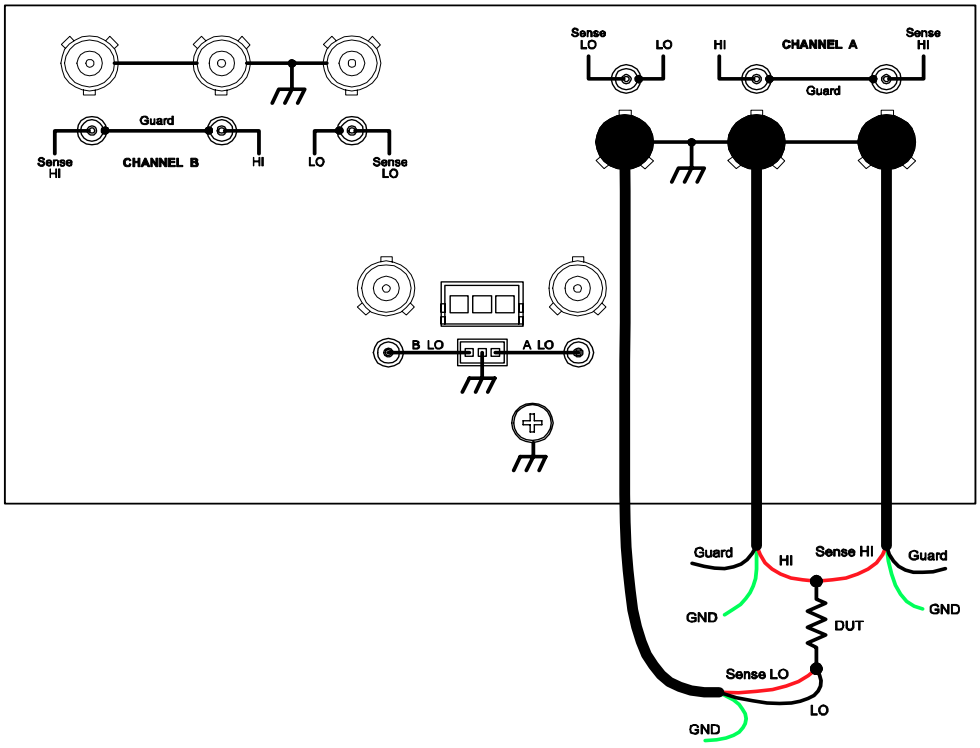
Use 4-wire remote sensing for the following source-measure conditions:

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

Figure 3-6
Model 2602/2612 four-wire connections (remote sensing)



Model 2636 four-wire connections (remote sensing)



Sense mode selection

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

Front panel sense selection

Table 3-1 summarizes the steps to check and/or change the sense mode front panel. When in the menu structure, use the rotary knob (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 2601/2611/2635	Model 2602/2612/2636
1) Press **CONFIG** key 2) Select **SRC** or **MEAS** menu* 3) Select **V-SOURCE** menu 4) Select **SENSE-MODE** menu 5) Select **2-WIRE** or **4-WIRE**	1) Press **CONFIG** key 2) Select **CHANNEL-A** or **CHANNEL-B** 3) Select **SRC** or **MEAS** menu* 4) Select **V-SOURCE** menu 5) Select **SENSE-MODE** menu 6) Select **2-WIRE** or **4-WIRE**

* 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
smuX.source.output = smuX.OUTPUT_OFF	Turns off the SMU output.
smuX.sense = smuX.SENSE_LOCAL	Selects local (2-wire) sense.
smuX.sense = smuX.SENSE_REMOTE	Selects remote (4-wire) sense.

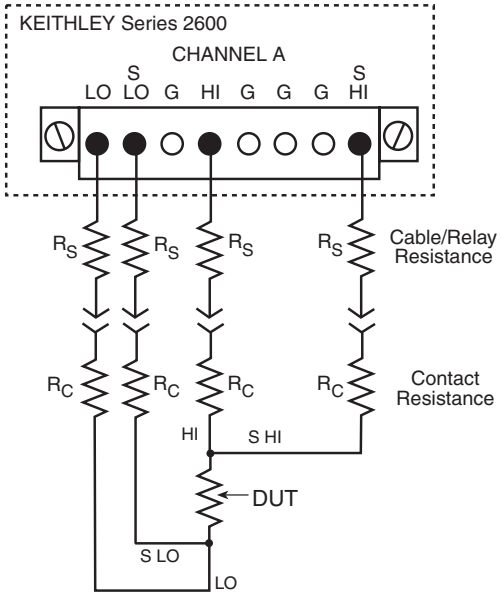
* Model 2601/2611/2635: smuX = smua. Model 2602/2612/2636: smuX = smua (Channel A) or smub (Channel B).

Contact check connections

The contact check function[1] prevents measurement errors due to excessive resistance in the force or sense leads. Connections for contact check measurements are shown in Figure 3-7. See Section 4 for operation and Section 12 for details on contact check commands.

1. All Model 2611/2612 System SourceMeters manufactured by Keithley Instruments support the contact check function. Models 2635 and 2636 do not support the contact check function. Only Models 2601/2602 with firmware Revision 1.1.0 or later and source measure unit (SMU) hardware Revision E or later support the contact check function. To determine the firmware and SMU hardware revisions, inspect the data returned by the print(localnode.info()) command. The InstFwRev and SMUBrdRev keys contain the necessary information.

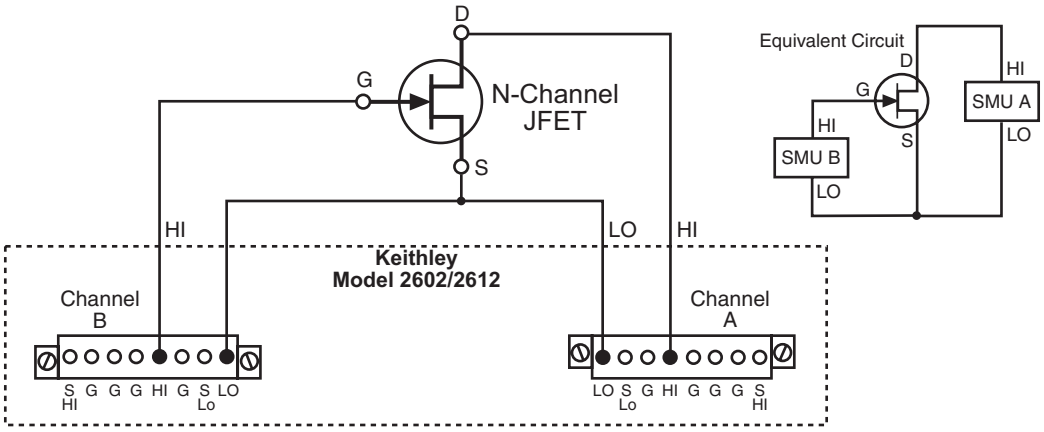
Figure 3-7
Contact check connections



Multiple SMU connections

Figure 3-8 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 3-8
Model 2602/2612 two SMUs connected to a 3-terminal device (local sensing)



Model 2636, two SMUs connected to a 3-terminal device (local sensing, floating)

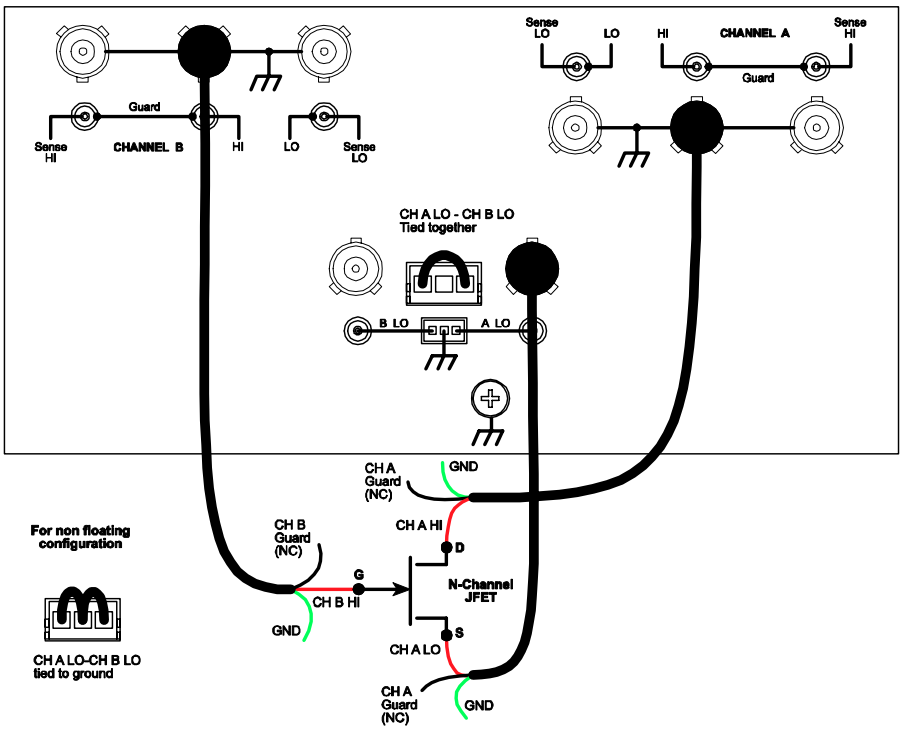
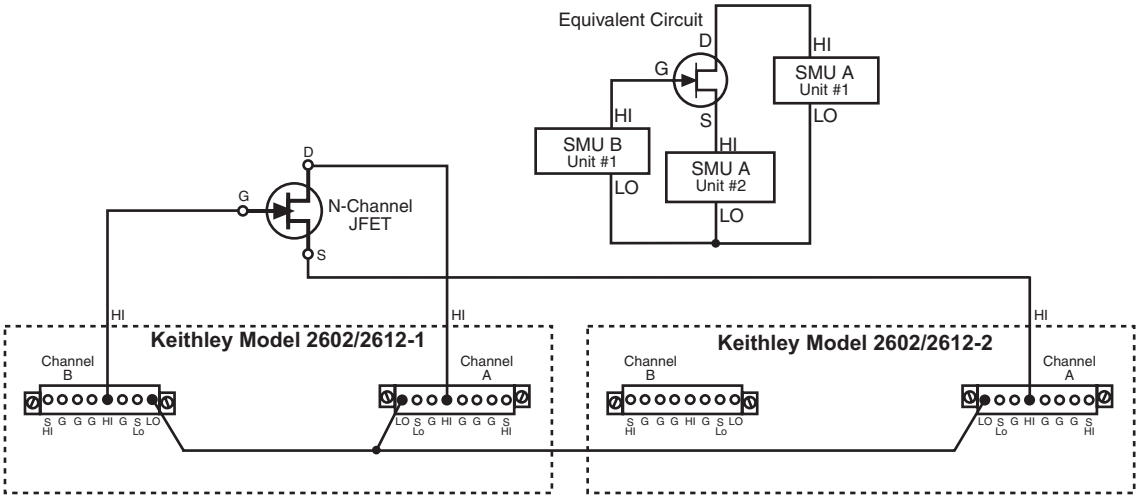
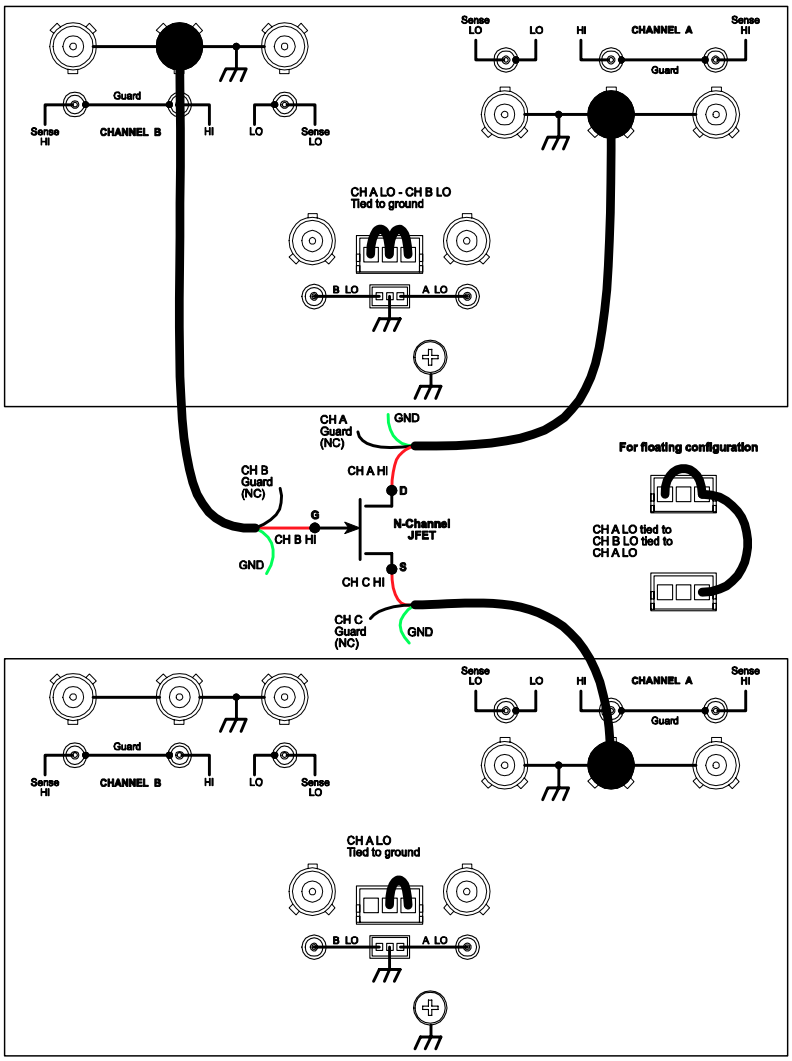


Figure 3-9 shows how to use three SMUs to test the same 3-terminal device. The third SMU is connected to the source (S) terminal of the JFET. This allows the source terminal to be biased above signal low. Setting this SMU to output 0V effectively connects the source terminal of the JFET to signal low.

Figure 3-9
Three SMUs connected to a 3-terminal device



Model 2636, three SMUs connected to a 3-terminal device (local sensing, non-floating)



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 3-10) should be used for the following source-measure condition:

- Test circuit impedance is >1GΩ.

NOTE See "Guarding and shielding" in Section 8 for details on the principles of guarding.

Figure 3-10
Models 2602 and 2612 high-impedance guarding

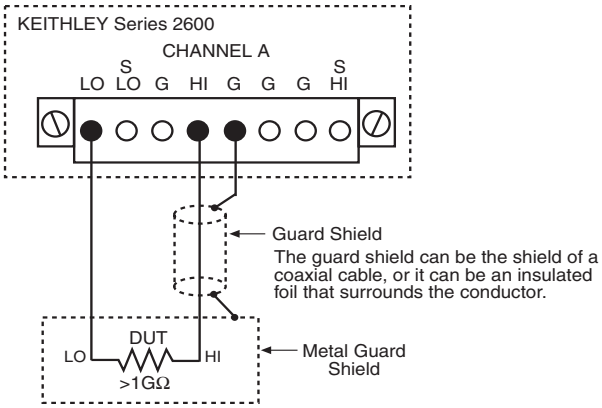


Figure 3-11
Model 2636 high-impedance guarding (floating)

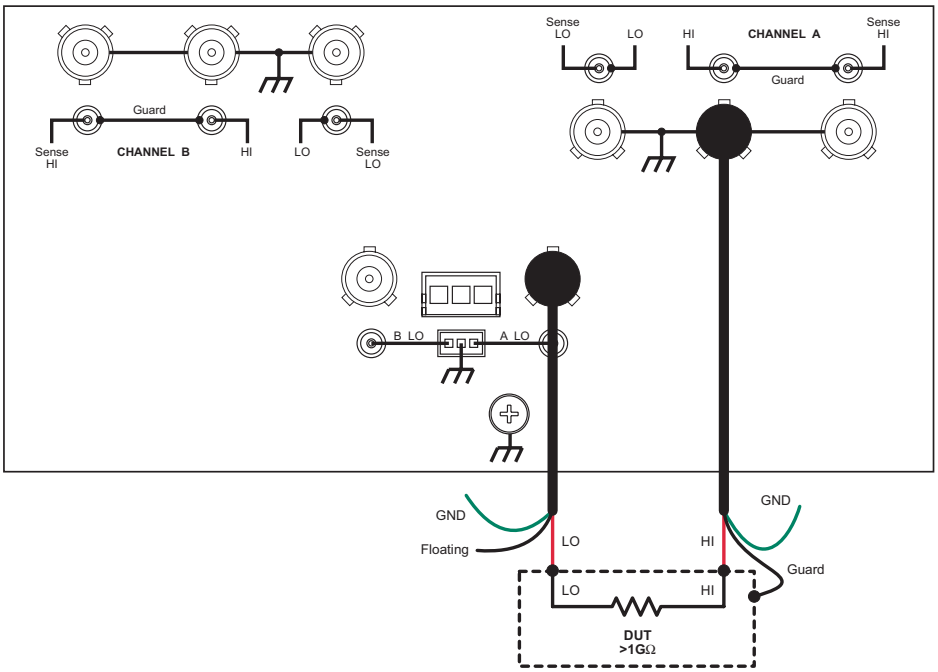
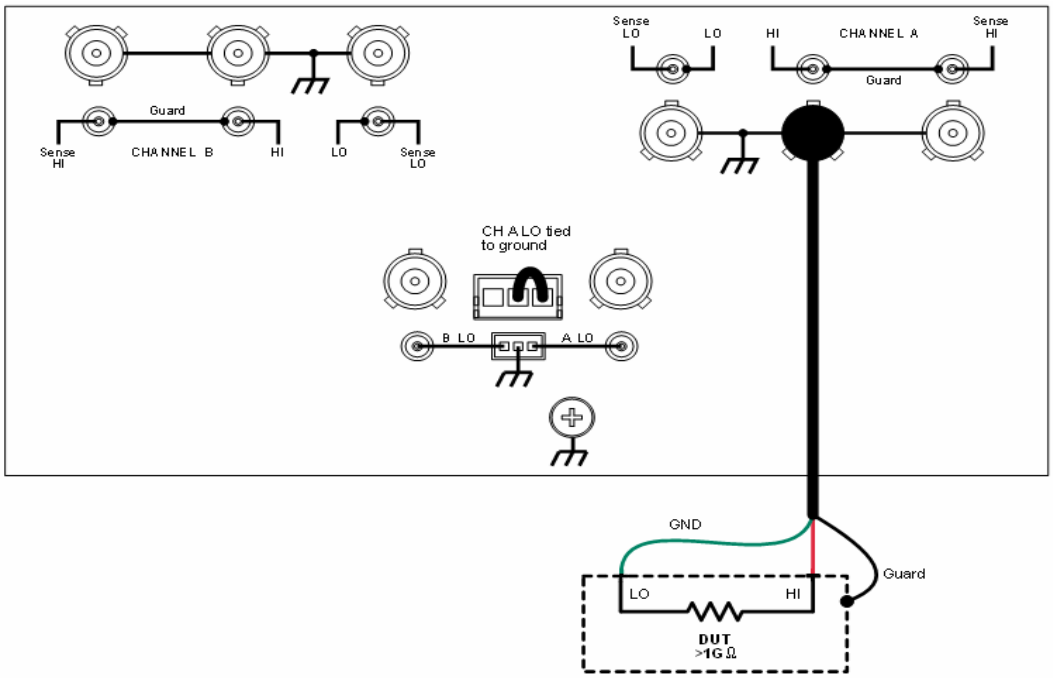


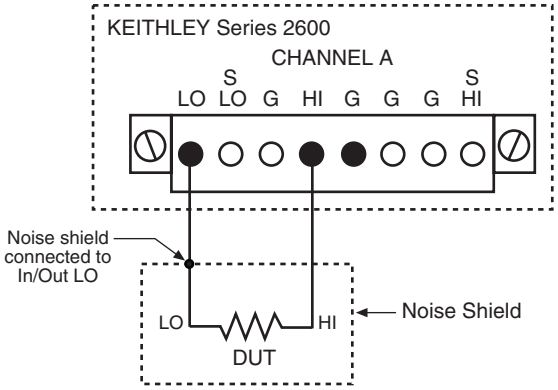
Figure 3-12
Model 2636 High-impedance guarding (non-floating)



Noise shield

A noise shield (see Figure 3-13) 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 3-13.

Figure 3-13
Models 2602 and 2612 noise shield



Model 2636 noise shield (non-floating)

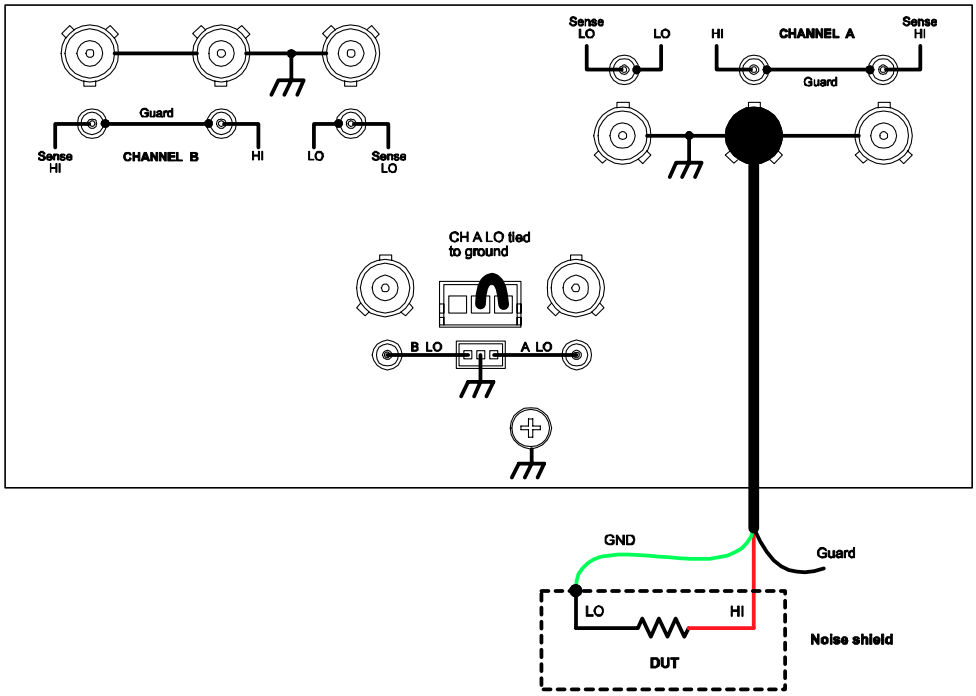


Figure 3-14
Model 2636 noise shield (non-floating)

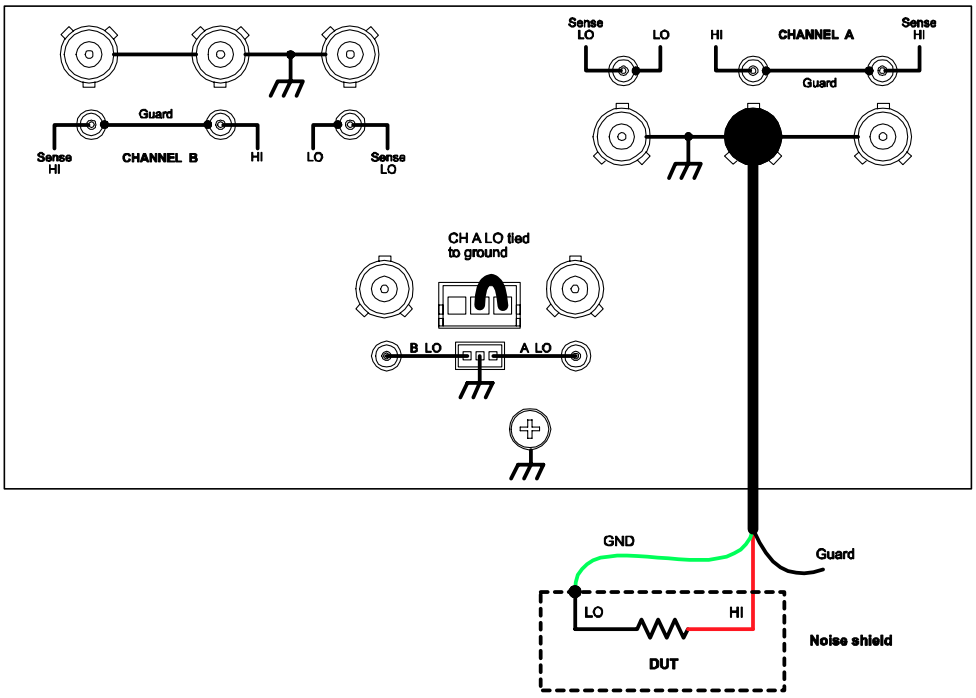
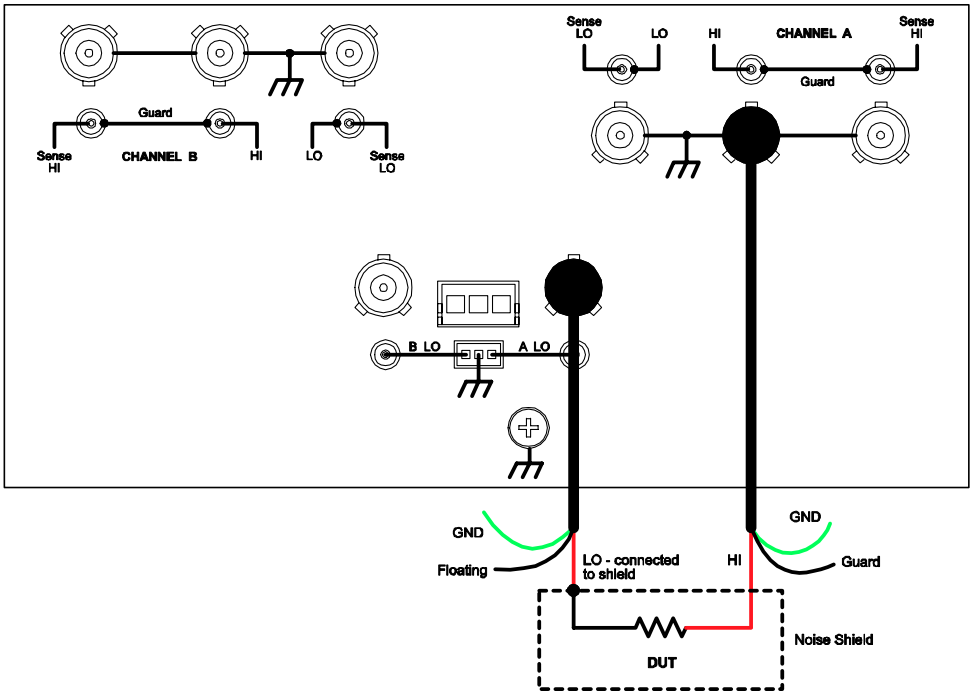


Figure 3-15
Model 2636 noise shield (floating)



Safety shield

A safety shield must be used whenever hazardous voltages (>30Vrms, 42Vpeak) 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 the section "Test fixture" later in this section for important safety information on the use of a metal or nonmetallic enclosure.

Model 2601/2602 safety shield

The maximum output voltage for a Model 2601/2602 channel is 40V, which is considered a non-hazardous level. However, using two or more Model 2601/2602 voltage sources in a series configuration can cause test circuit voltage to exceed 42V. For example, the SMUs of two Model 2601/2602 instruments can be connected in series to apply 80V to a DUT (see Figure 3-16).

The connections for the test configuration in Figure 3-16 are shown in Figure 3-17. 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 a SMU" later in this section).

Figure 3-16
Safety shield for hazardous voltage using two 2601/2602 channels (>42V)

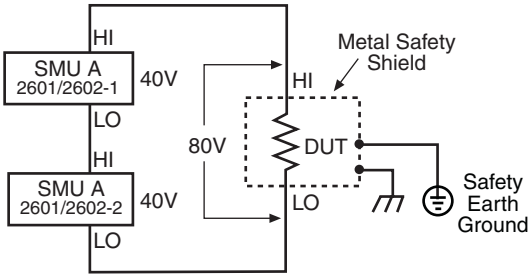
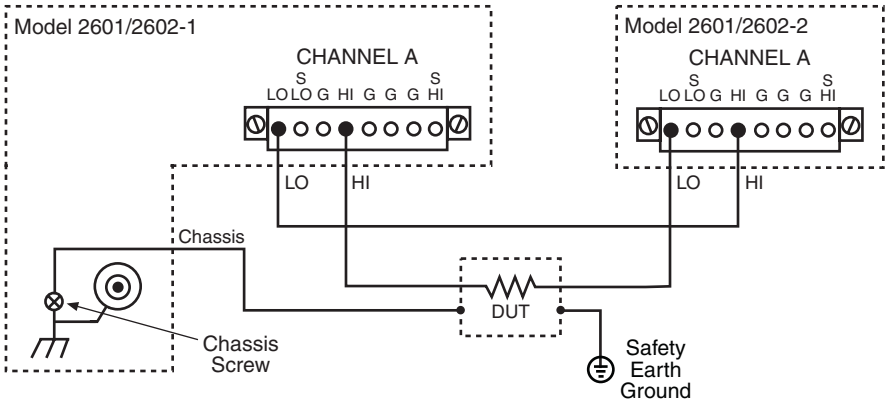


Figure 3-17
Model 2601/2602-1 connections for test circuit shown in Figure 3-16



Model 2611/2612/2635/2636 safety shield

The maximum output voltage for a Model 2611/2612/2635/2636 channel is 200V, which is considered hazardous and requires a safety shield (Figure 3-18). The connections for the test configuration in Figure 3-18 are shown in Figure 3-20. Use #18AWG wire or larger for connections to safety earth ground and chassis.

Figure 3-18
Safety shield for Models 2611/2612/2635/2636 hazardous voltage (200V maximum)

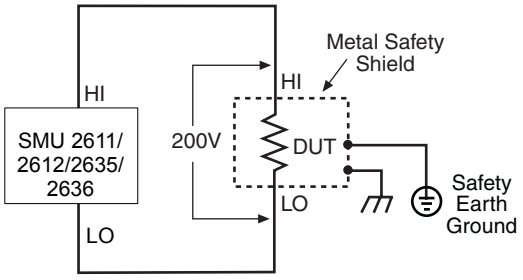


Figure 3-19
Model 2601/2602-1 connections for test circuit shown in Figure 3-18

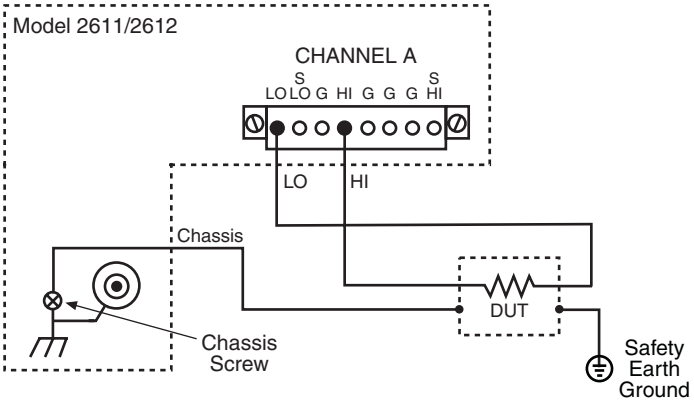
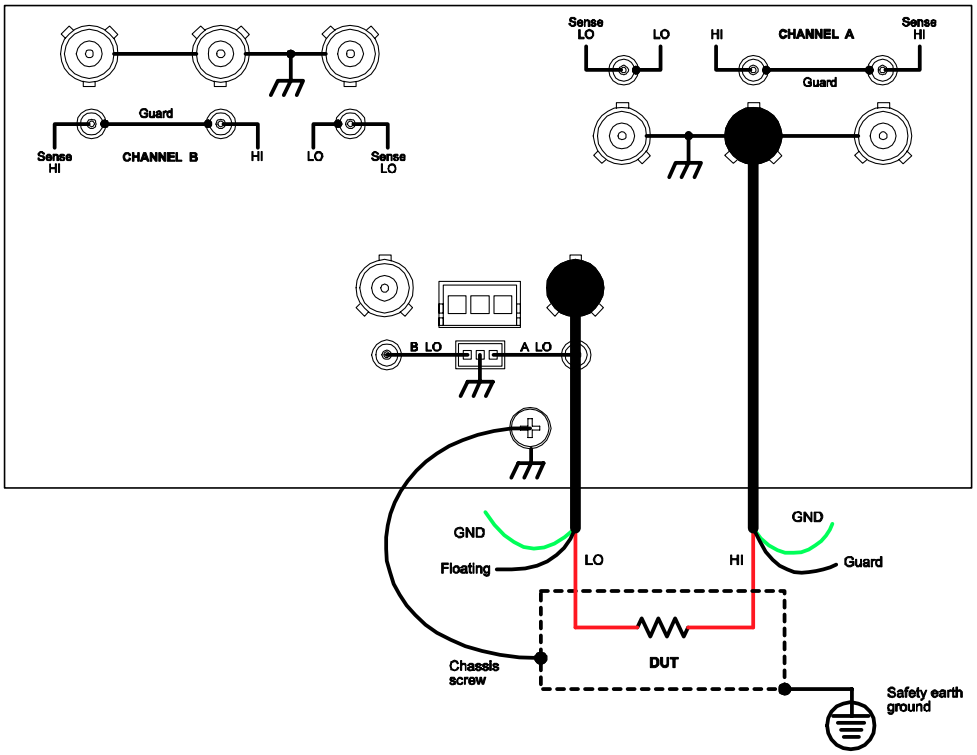


Figure 3-20
Model 2636 connections for test circuit shown in Figure 3-18



Using shielding and guarding together

Figure 3-21 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 3-21
Model 2601/2602-1 connections for noise shield, safety shield, and guarding

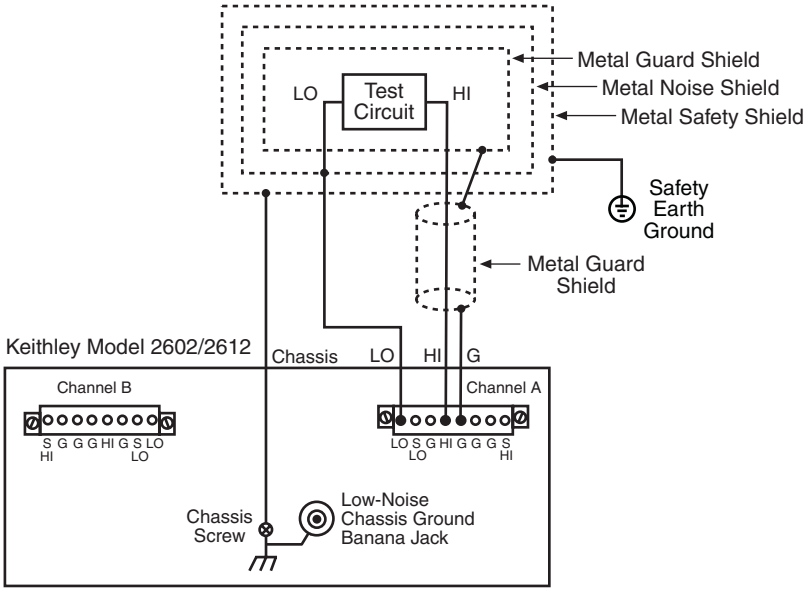
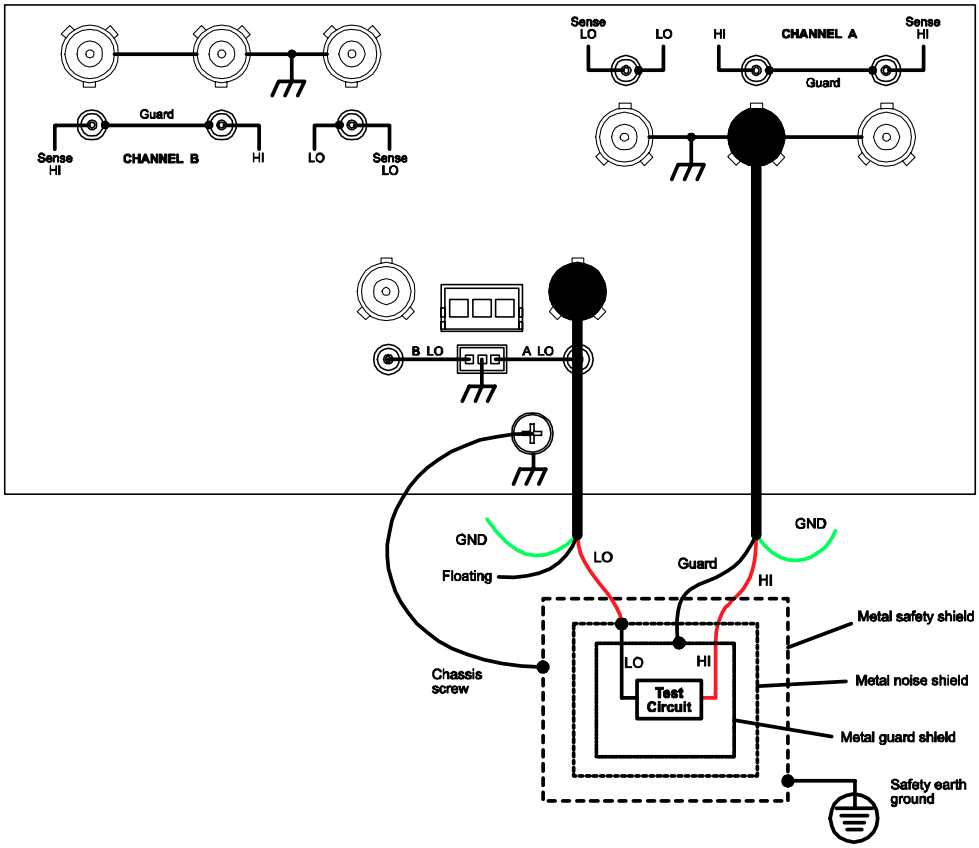


Figure 3-22
Model 2636 connections for noise shield, safety shield, and guarding



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 (>30Vrms, 42Vpeak) 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 above**.

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

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

WARNING ***When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to reliably protect the operator from exposed voltages.***

The output enable pin on the ***digital I/O port on the*** *Models 2601 and2602 SourceMeter is not suitable for control of safety circuits and should not be used to control a safety interlock. The Interlock pin on the* ***digital I/O port*** *for the Models 2611, 2612, 2635, and 2636 can be used to control a safety interlock.*

Floating a SMU

Using an external source in the test system may require that a Series 2600 SMU float off chassis earth ground. An example of such a test system is shown in Figure 3-23, 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 3-23, the Series 2600 must float off chassis earth ground. As shown, SMU LO of the Model 26xx is floating +10V above chassis earth ground. If SMU LO of the Model 26xx was instead connected to chassis ground, the external voltage source would be shorted through chassis ground.

The Series 2600 connections for the floating configuration (Figure 3-23) are shown in Figure 3-24. 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 3-23 and Figure 3-24 can instead be a SMU of a second Series 2600 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 a SMU is ±250V. Exceeding this level may cause damage to the instrument and create a shock hazard.***

Using an external source to float a SMU could create a shock hazard in the test circuit. A shock hazard exists whenever >42V 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 >42V 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" earlier in this section).

Figure 3-23
Floating the Series 2600

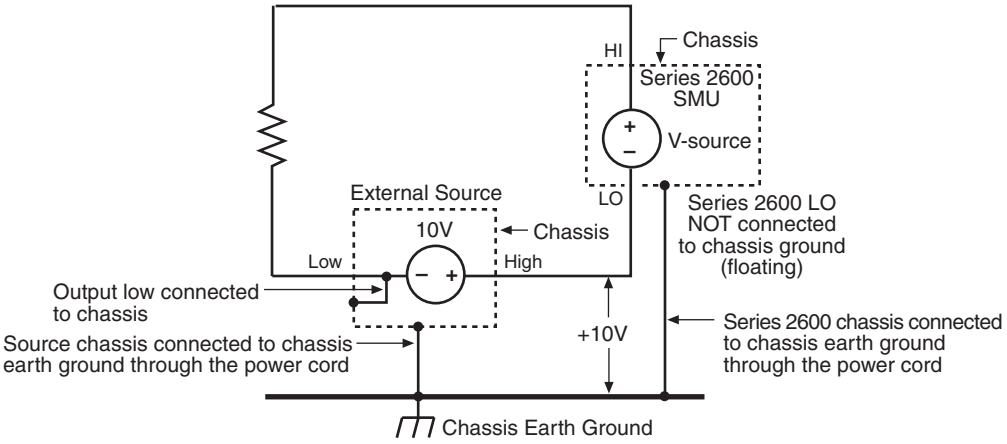
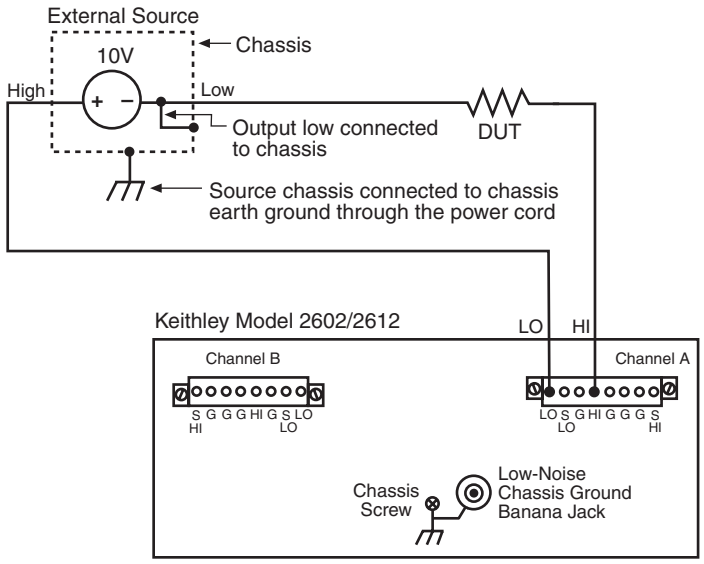
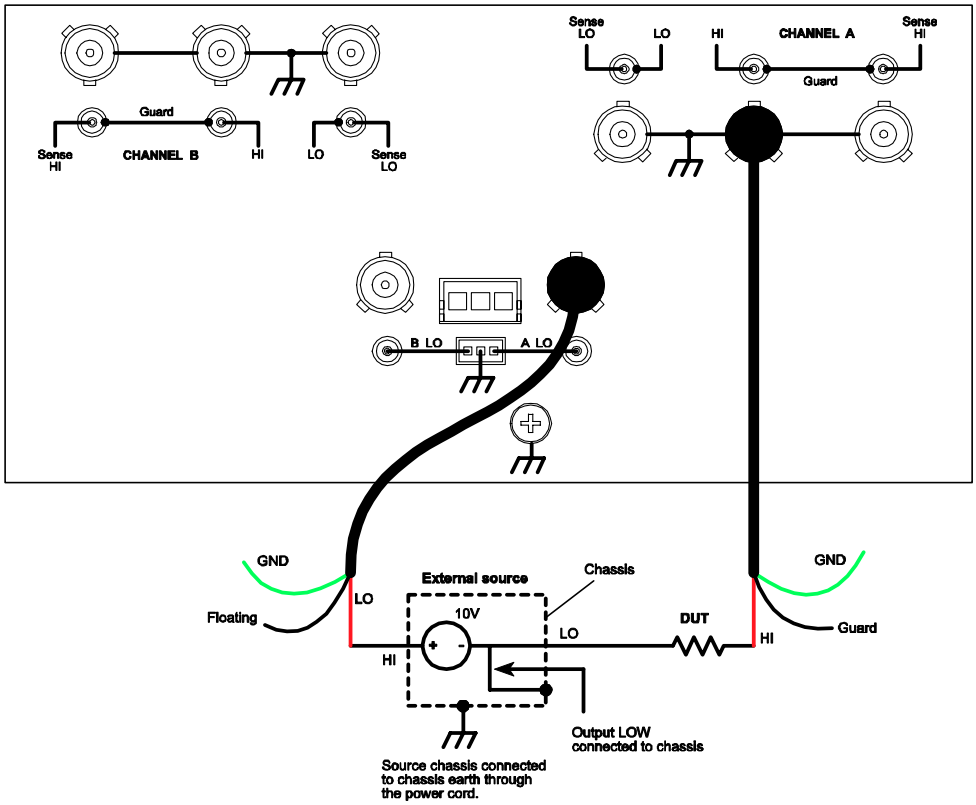


Figure 3-24
Model 2601/2602-1 SMU connections for the floating configuration shown in Figure 3-23



Model 2636 SMU connections for the floating configuration shown in Figure 3-23



Output-off states

When a SMU is turned off, it may not be completely isolated from the external circuit that it is connected to. There are three output-off states for a Series 2600 SMU: Normal, High Impedance or zero. For the Models 2602, 2612, and 2636, each SMU channel 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. When running firmware version earlier than 1.2.0, current compliance will set to 10% full scale of the present current range, or 100µA (whichever is smaller). When running firmware version 1.2.0 or later, current compliance determined by the `smuX.source.offlimiti` command (default 1mA) will be used. 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.offlimiti = ivalue	Sets current limit in normal output-off state for firmware revision 1.2.0, or later.
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 2601/2611/2635: smuX = smua, Model 2602/2612/2636: smuX = smua (Channel A) or smub (Channel B).

Section 4

Basic Operation

In this section:

Topic	Page
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	**4-6**
Warm-up	4-6
Auto zero	4-6
NPLC caching	4-6
Basic source-measure procedure	**4-8**
Front panel source-measure procedure	4-8
Remote source-measure procedure	4-9
Measure only	**4-11**
Sink operation	**4-12**
Ohms measurements	**4-12**
Ohms calculations	4-12
Ohms ranging	4-12
Basic ohms measurement procedure	4-12
Ohms sensing	4-13
Sense selection	4-14
Remote ohms programming	4-15
Power measurements	**4-16**
Power calculations	4-16
Basic power measurement procedure	4-16
Remote power programming	4-16
Contact check measurements	**4-17**
Overview	4-17
Contact check commands	4-18
Contact check programming example	4-19

Overview

The documentation in this section provides basic operating instructions for the Keithley Instruments Series 2600 System SourceMeter® and includes the following:

- "Operation overview"
- "Operation considerations"
- "Measure only"
- "Sink operation"
- "Ohms measurements"
- "Power measurements"
- "Contact check measurements"

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" in Section 8.

NOTE See the specifications in Appendix A for important performance aspects that may affect measurements.

Table 4-1
Source-measure capabilities

Model 2601/2602			Model 2611/2612			Model 2635/2636		
Range	**Source**	**Measure**	**Range**	**Source**	**Measure**	**Range**	**Source**	**Measure**
100mV	±101mV	±102mV	200mV	±202mV	±204mV	200mV	+/-202mV	+/-204mV
1V	±1.01V	±1.02V	2V	±2.02V	±2.04V	2V	+/-2.02V	+/-2.04V
6V	±6.06V	±6.12V	20V	±20.2V	±20.4V	20V	+/-20.2V	+/-20.4V
40V	±40.4V	±40.8V	200V[1]	±202V	±204V	200V[3]	+/-202V	+/-204V
100nA	±101nA	±102nA	100nA	±101nA	±102nA	100pA	N/A	+/-102pA
1µA	±1.01µA	±1.02µA	1µA	±1.01µA	±1.02µA	1nA	+/-1.01nA	+/-1.02nA
10µA	±10.1µA	±10.2µA	10µA	±10.1µA	±10.2µA	10nA	+/-10.1nA	+/-10.2nA
100µA	±101µA	±102µA	100µA	±101µA	±102µA	100nA	+/-101nA	+/-102nA
1mA	±1.01mA	±1.02mA	1mA	±1.01mA	±1.02mA	1µA	±1.01µA	±1.02µA
10mA	±10.1mA	±10.2mA	10mA	±10.1mA	±10.2mA	10µA	±10.1µA	±10.2µA
100mA	±101mA	±102mA	100mA	±101mA	±102mA	100µA	±101µA	±102µA
1A	±1.01A	±1.02A	1A	±1.01A	±1.02A	1mA	±1.01mA	±1.02mA
3A	±3.03A	±3.06A	1.5A	±1.515A	±1.53A	10mA	±10.1mA	±10.2mA
			10A[2]	±10.1A	±10.2A	100mA	±101mA	±102mA
						1A	±1.01A	±1.02A
						1.5A	±1.515A	±1.53A
Max Power = 40.4W per channel			Max Power = 30.603W per channel 1. 200V source range available only when interlock is enabled. See Section 10. 2. 10A range available only in pulse mode.			Max Power = 30.603W per channel 3. 200V source range available only when interlock is enabled. See Section 10.		

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" in Section 8 for more information.

NOTE The only exception to the compliance limit not being exceeded is the VLIMIT when operating as an ISOURCE. To avoid excessive (and potentially destructive) currents from flowing, the VLIMIT will source or sink up to 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

Model 2601/2602		Model 2611/2612		Model 2635/2636	
Source range	**Maximum compliance value**	**Source range**	**Maximum compliance value**	**Source range**	**Maximum compliance value**
100mV 1V 6V 40V	3A 3A 3A 1A	200mV 2V 20V 200V	1.5A 1.5A 1.5A 100mA	200mV 2V 20V 200V	1.5A 1.5A 1.5A 100mA
100nA 1µA 10µA 100µA 1mA 10mA 100mA 1A 3A	40V 40V 40V 40V 40V 40V 40V 40V 6V	100nA 1µA 10µA 100µA 1mA 10mA 100mA 1A 1.5A	200V 200V 200V 200V 200V 200V 200V 20V 20V	1nA 10nA 100nA 1µA 10µA 100µA 1mA 10mA 100mA 1A 1.5A	200V 200V 200V 200V 200V 200V 200V 200V 200V 20V 20V

*smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.

Setting the compliance limit

Front panel compliance limit

Set the compliance limit from the front panel as follows:

1. For the Model 2601/2611/2635 or the Model 2602/2612/2636 single-channel display mode, press the LIMIT key to directly access compliance editing.
2. For the Model 2602/2612/2636 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

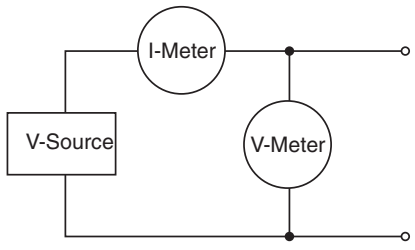
Command*	Description
smuX.source.limiti = limit smuX.source.limitv = limit compliance = smuX.source.compliance	Set current compliance limit. Set voltage compliance limit. Test if in compliance (1 = in compliance; 0 = not in compliance).

*smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.

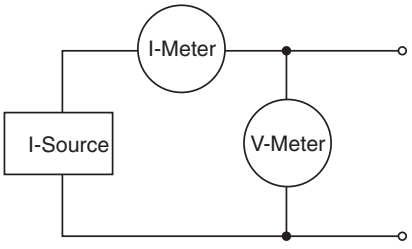
Basic circuit configurations

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" in Section 8 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 Series 2600 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

Auto zero setting	Description
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 4 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[1]	Description
smuX.measure.autozero = smuX.AUTOZERO_OFF smuX.measure.autozero = smuX.AUTOZERO_ONCE smuX.measure.autozero = smuX.AUTOZERO_AUTO	Disable auto zero.[2] Force one ref and zero. Force ref and zero with each measurement.

*smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.

**Old NPLC cache values will be used when auto zero is disabled. See "NPLC caching" earlier in this section.

Basic source-measure procedure

Front panel source-measure procedure

Use the following procedure to perform the basic source-measure operations of the Series 2600 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 Series 2600 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 2601/2611/2635 or the Model 2602/2612/2636 single-channel display mode, press the LIMIT key.
2. For the Model 2602/2612/2636 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 2602/2612/2636 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 table summarizes the basic commands and gives a simple programming example.

Basic source-measure commands

Table 4-6 summarizes basic source-measure commands. See Section 12 for more information on using these commands.

Table 4-6
Basic source-measure commands

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

* smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.

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

```
smua.reset()  --Restore Series 2600 defaults.

smua.source.func = smua.OUTPUT_DCVOLTS  --Select voltage
   source function.

smua.source.autorangev = smua.AUTORANGE_ON  --Set source
   range to auto.
```

```
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" described earlier in this section 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, 0.00000nA for Models 2635/2636).
   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" earlier in this section for details.

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

The sink operating limits are shown in "General SourceMeter power equation" in Section 8.

Ohms measurements

Ohms calculations

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

$R = V/I$

Where: R is the calculated resistance

V is the measured voltage

I is the sourced current

Ohms ranging

The front panel ohms function does not use ranging. The unit formats a calculated V/I reading to best fit the display. There may be leading zeros if the ohms reading is very small (<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 Series 2600 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.***

To take an ohms measurement:

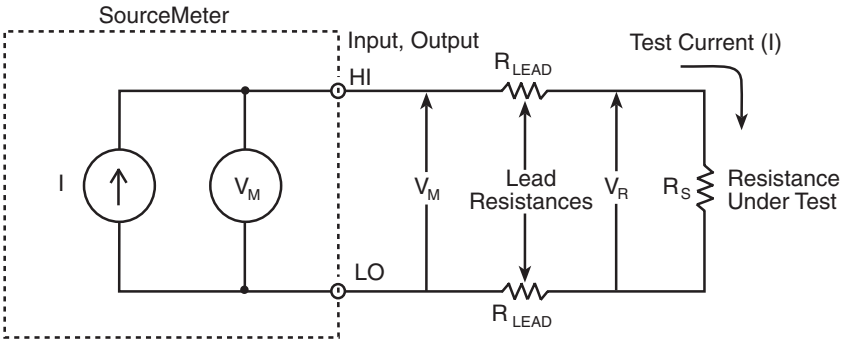
1. For the Model 2602/2612/2636, press the DISPLAY key to select the single-channel display mode.
2. Press SRC to select the current source function, then set the output current to the desired value based on the expected resistance. See Step 1 of "Front panel source-measure procedure" earlier in this section.
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" earlier in this section.
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 4-2, 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 4-3 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 4-2
2-wire resistance sensing

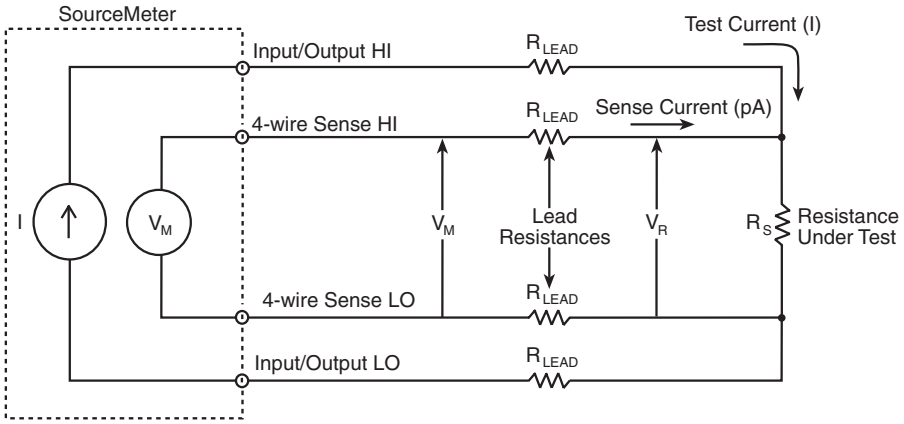


I = Current sourced by SourceMeter
V_M = Voltage measured by SourceMeter
V_R = Voltage across resistor

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

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

Figure 4-3
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, $V_M = V_R$

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

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 4-6 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 4-6 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 Series 2600 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 12.

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 Series 2600 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 2602/2612/2636, 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" described earlier in this section.
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".
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 4-6 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, auto source range, 5V output
- Current measure function and range: current, auto
- Current compliance: 50mA

```
smua.reset()  --Restore Series 2600 defaults.

smua.source.func = smua.OUTPUT_DCVOLTS  --Select voltage
   source function.

smua.source.autorangev = smua.AUTORANGE_ON  --Set source
   range to auto.

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.
```

Contact check measurements

Overview

The contact check function[1] prevents measurements that may be in error due to excessive resistance in the force or sense leads when making remotely sensed (Kelvin) measurements. Potential sources for this resistance include poor contact at the DUT, failing relay contacts on a switching card, and wires that are too long or thin. The contact check function will also detect an open circuit that may occur with a four-point probe is misplaced or misaligned. This relationship is shown schematically in Figure 4-4, where R_C is the resistance of the mechanical contact at the DUT, and R_S is the series resistance of relays and cables.

Models 2601/2602/2611/2612 have the contact check function. Models 2635/2636 do not have the contact check function.

1. All Model 2611/2612 System SourceMeters manufactured by Keithley Instruments support the contact check function. Models 2635 and 2636 do not support the contact check function. Only Models 2601/2602 with firmware Revision 1.1.0 or later and source measure unit (SMU) hardware Revision E or later support the contact check function. To determine the firmware and SMU hardware revisions, inspect the data returned by the print(localnode.info()) command. The InstFwRev and SMUBrdRev keys contain the necessary information.

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 Series 2600 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.***

Contact check commands

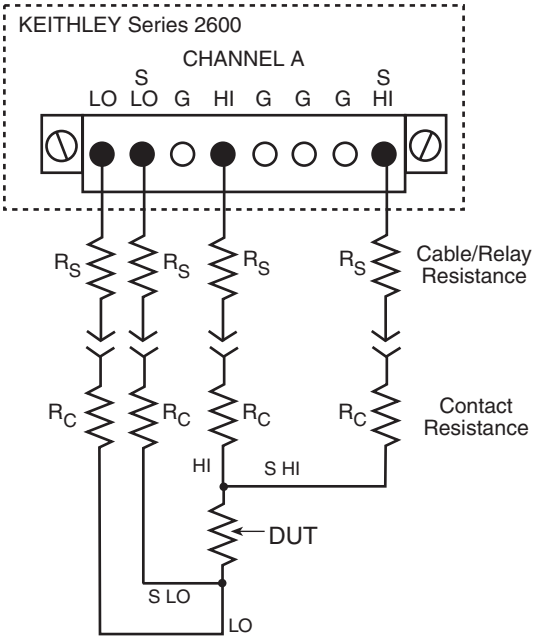
Table 4-7 summarizes basic contact check commands. See Section 12 for more information on using these commands.

Table 4-7
Basic contact check commands

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

*smuX = smua for the Model 2601/2611; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612.

Figure 4-4
Contact check measurements



Contact check programming example

The command sequence for a typical contact measurement is shown below. These commands set the contact check speed to fast and the threshold to 10Ω. A contact check measurement against the threshold is then made. If it fails, a more accurate contact check measurement is made, and the test is aborted. Otherwise, the output is turned on, and the test continues.

```
smua.reset()                                    - Restore defaults.
smua.contact.speed = smua.CONTACT_FAST          - Set speed to fast.
smua.contact.threshold = 10                     - Set threshold to 10Ω.

if (not smua.contact.check()) then              - Check contacts against threshold.
    --One/both contact resistances are not
    --below the programmed threshold.
    smua.contact.speed = smua.CONTACT_SLOW      - Set speed to slow.
    rhi, rlo = smua.contact.r()                 - Get resistance readings.
    print(rhi, rlo)                             - Return contact resistances to the host.
    exit()                                        Terminate execution.
end
smua.source.output = smua.OUTPUT_ON             - Turn on output and continue.
```

This page left blank intentionally.

Section 5

Sweep Operation

In this section:

Topic	Page
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	5-6
Sweep measurement storage	5-7
Sweep functions	**5-7**
Staircase sweep functions	5-8
Pulse sweep functions	5-8
Custom sweep functions	5-9
Running sweeps	**5-9**
Front panel	5-9
Sweep programming examples	5-9

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"
- "Sweep characteristics"
- "Sweep functions"
- "Running sweeps"

Sweep overview

As shown in Figure 5-1, the Keithley Instruments Series 2600 System 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.

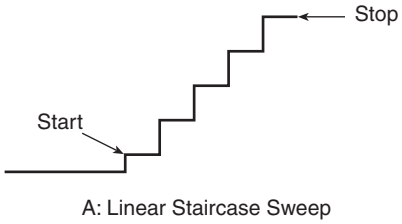
Logarithmic staircase sweep – In this case, the current or voltage increases or decreases logarithmically, beginning with a start voltage or current and ending with a stop voltage or current. Figure 5-1B shows an increasing logarithmic staircase sweep.

Custom (list) sweep – The custom sweep allows you to program arbitrary sweep steps anywhere within the output voltage or current range of the Series 2600. Figure 5-1C shows a typical custom sweep with arbitrary steps.

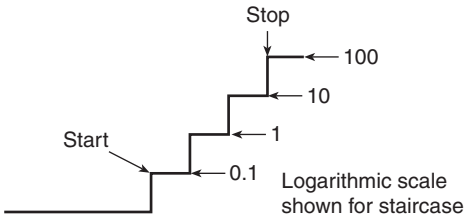
Pulse sweeps – Two sweeps can also be performed as a pulse sweep: fixed current pulse or fixed voltage pulse. These pulse sweeps are similar to those outlined above except that a pulse of a specific width is generated at each point instead of a constant level.

NOTE The Pulse Sweep information provided in this section applies to the pulse functions of the KIGeneral factory script. For firmware revisions 1.2.0 and later, significantly enhanced pulse capabilities are available with the KIPulse factory script. Refer to Section 13 for more information on both the KIGeneral and KIPulse factory scripts.

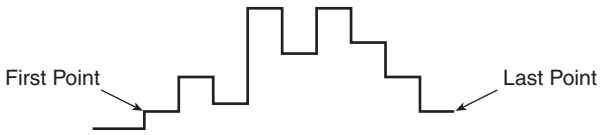
Figure 5-1
Comparison of staircase sweep types



A: Linear Staircase Sweep



B: Logarithmic Staircase Sweep



C: Custom Sweep

Sweep characteristics

Linear staircase sweeps

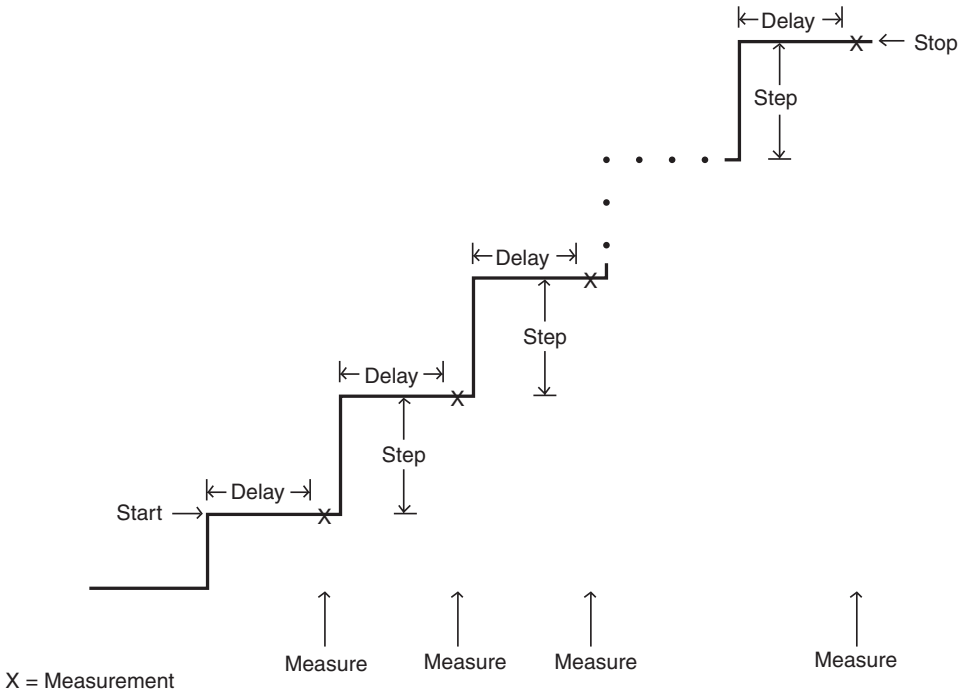
As shown in Figure 5-2, this sweep type steps from a start voltage or current value to an ending (stop) value. A measurement is made at each step after a specified delay period (settling time). Programmable parameters include the source function, channel, start and stop levels, the number of sweep points, 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:

step = (stop - start) / (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.

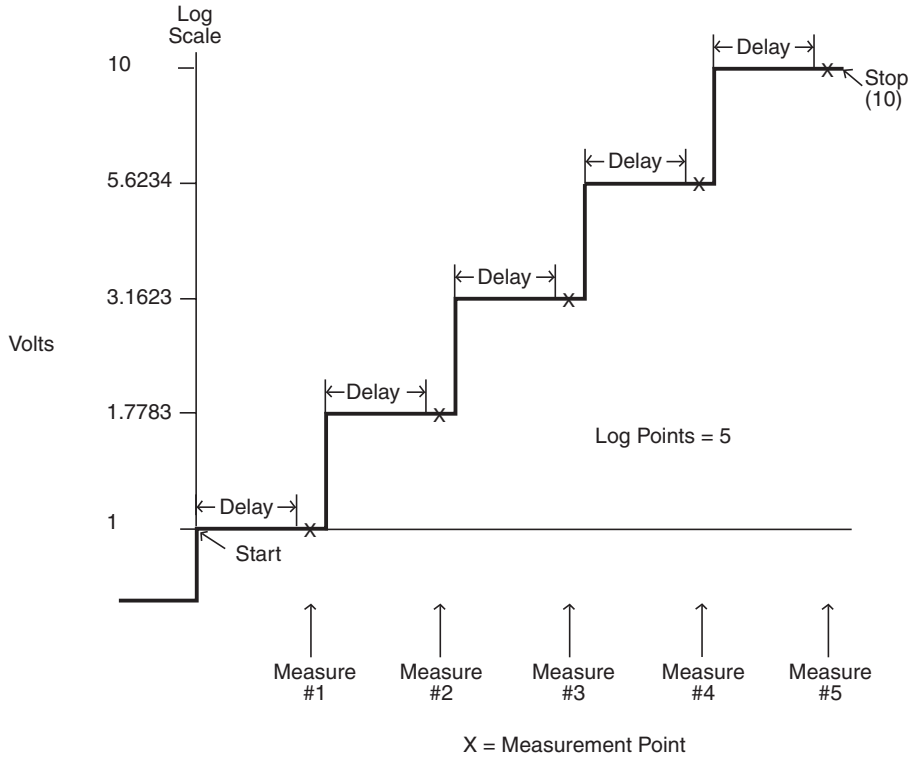
Figure 5-2
Linear staircase sweep



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:

$$\text{Log Step Size} = \frac{\log10(\text{stop}) - \log10(\text{start})}{\text{Points} - 1}$$
$$= \frac{\log10(10) - \log10(1)}{5 - 1}$$
$$= \frac{(1 - 0)}{4}$$
$$= 0.25$$

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.

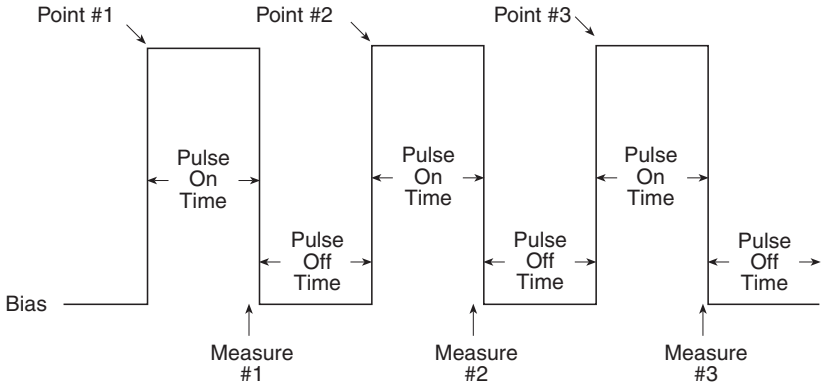
Pulse sweeps

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. Refer to Section 13 for more KIPulse information and more detailed information on KIGeneral.

Figure 5-4
Pulse sweep example



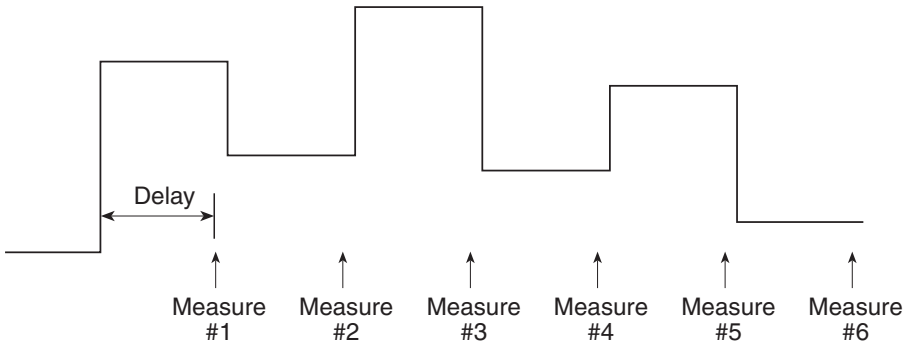
Custom (list) sweeps

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 Visit 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

`SweepILinMeasureV(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 (≥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 (≥2).

`SweepILogMeasureV(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 (≥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 (≥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 (≥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 (≥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.

Table 5-5
Sweep example parameters

Sweep type	Parameters for sweep examples
Linear staircase sweep	Start current: 1mA Stop current: 10mA # points: 10 Settling time: 0.1s
Pulse current sweep	Bias current: 1mA On current: 10mA Pulse on time: 10ms Pulse off time: 50ms # points: 10
Custom (list) sweep	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 Series 2600 defaults.
   smua.source.limitv = 1          -- Set compliance to 1V.
   ```

2. **Configure and execute 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, 10e-3, 0.1, 10)
   --Linear staircase sweep, Channel A, 1mA to 10mA, 0.1 second delay, 10 points.

   waitcomplete()
   -- Wait for sweep to complete.
   ```

3. **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 Series 2600 defaults.
   smua.source.limitv = 10         --Set compliance to 5V.
   ```

2. **Configure and execute 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, 10e-3, 10e-3, 50e-3, 10)
   -- Pulse current sweep, Channel A, 1mA bias, 10mA level, 10ms pulse on, 50ms
   pulse off, 10 cycles.

   waitcomplete()
   -- Wait for sweep to complete.
   ```

3. **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 Series 2600 defaults.
   smua.source.limiti = 10e-3      -- Set compliance to 10mA.
   ```

2. **Configure and execute the sweep.**

   Example – The following parameters configure a list sweep with 3V, 1V, 4V, 5V, 2V points using a 0.1s settling time:

   ```
   vlist = {3, 1, 4, 5, 2}
    -- Define voltage list.
   SweepVListMeasureI(smua, vlist, 0.1, 5)
    -- List sweep, channel A, 3V, 1V, 4V, 5V, 2V steps, 0.1s delay, 5 points.
   waitcomplete()
     -- Wait for sweep to complete.
   ```

3. **Request readings.**

   Request readings from Buffer 1 as follows:

   ```
   printbuffer(1, 5, smua.nvbuffer1.readings)
   ```

This page left blank intentionally.

Section 6

Range, Digits, Speed, Rel, and Filters

In this section:

Topic	Page
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	6-3
Range considerations	6-4
Range programming	6-4
Digits	**6-6**
Setting display resolution	6-6
Remote digits programming	6-6
Speed	**6-6**
Setting speed	6-7
Remote speed programming	6-7
Rel	**6-8**
Front panel rel	6-8
Remote rel programming	6-9
Filters	**6-10**
Filter types	6-10
Response time considerations	6-10
Front panel filter control	6-10
Remote filter programming	6-13

Overview

The documentation in this section provides detailed information on characteristics and script programming for each of the following functions:

- "Range"
- "Digits"
- "Speed"
- "Rel"
- "Filters"

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., --.---- µA), 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 Keithley Instruments Series 2600 System SourceMeters®.

Table 6-1
Source and measurement ranges

Model 2601/2602		**Model 2611/2612**		**Model 2635/2636**	
Voltage Ranges	**Current Ranges**	**Voltage Ranges**	**Current Ranges**	**Voltage Ranges**	**Current Ranges**
100mV 1V 6V 40V	100nA 1µA 10µA 100µA 1mA 10mA 100mA 1A 3A	200mV 2V 20V 200V	100nA 1µA 10µA 100µA 1mA 10mA 100mA 1A 1.5A 10A[1]	200mV 2V 20V 200V	100pA[2] 1nA 10nA 100nA 1µA 10µA 100µA 1mA 10mA 100mA 1A 1.5A

1. 10A range available only in pulse mode.
2. 100pA range only in measure.

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, ±1.01A is the full scale source value for the 1A range, and ±102mA 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

- Model 2601/2602: 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.
- Model 2611/2612/2636: With the 200V V-Source range selected, the highest current measurement range is 100mA. With I-Source ranges above 100mA selected, the highest voltage measurement range is 20V.
- 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 △ and ▽ 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 2602/2612 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.

Auto ranging

To use auto source ranging, press SRC then AUTO RANGE. To use auto measure ranging, select the Model 2602/2612/2636 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 Series 2600 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.

2601/2602 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 1A on the 100mA 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[1]	Description
Measure range commands: [2]	
smuX.measure.autorangei = smuX.AUTORANGE_ON	Enable current measure auto range.
smuX.measure.autorangei = smuX.AUTORANGE_OFF	Disable current measure auto range.
smuX.measure.autorangev = smuX.AUTORANGE_ON	Enable voltage measure auto range.
smuX.measure.autorangev = smuX.AUTORANGE_OFF	Disable voltage measure auto range.
smuX.measure.lowrangei = lowrange	Set lowest I measure range for auto range.
smuX.measure.lowrangev = lowrange	Set lowest V measure range for auto range.
smuX.measure.rangei = rangeval	Select manual current measure range.
smuX.measure.rangev = rangeval	Select manual voltage measure range.
Source range commands: [2]	
smuX.source.autorangei = smuX.AUTORANGE_ON	Enable current source auto range.
smuX.source.autorangei = smuX.AUTORANGE_OFF	Disable current source auto range.
smuX.source.autorangev = smuX.AUTORANGE_ON	Enable voltage source auto range.
smuX.source.autorangev = smuX.AUTORANGE_OFF	Disable voltage source auto range.
smuX.source.limiti = level	Set voltage source current limit.
smuX.source.limitv = level	Set current source voltage limit.
smuX.source.lowrangei = lowrange	Set lowest I source range for auto range.
smuX.source.lowrangev = lowrange	Set lowest V source range for auto range.
smuX.source.rangei = rangeval	Select manual current source range.
smuX.source.rangev = rangeval	Select manual voltage source range.

* smuX = smua for the Model 2601/2611; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612.

** See Table 6-1 for measure ranges.

*** See Table 6-1 for source 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: auto
- Current measure range: 10mA
- Measure low range: = 10µA
- Voltage source current limit: 10mA

```
smua.reset()
```
--Restore Series 2600 defaults.

```
smua.source.autorangev = smua.AUTORANGE_ON
```
--Set V source range to auto.

```
smua.measure.rangei = 1e-2
```
--Enable 10mA measure range.

```
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"” later in this section).

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 2602/2612/2636 dual-channel display mode, the maximum display resolution is 4.5 digits. For the Model 2602/2612/2636 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 summarizes digits commands. See Section 12 for more information.

Table 6-3
Digits commands

Command[1]	Description
display.smuX.digits = display.DIGITS_4_5 display.smuX.digits = display.DIGITS_5_5 display.smuX.digits = display.DIGITS_6_5	Set display to 4.5 digits. Set display to 5.5 digits. Set display to 6.5 digits.

1. smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.

Digits programming example

```
smua.reset()
```
--Restore Series 2600 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 5-1/2 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 2602/2612 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[1]	Description
smuX.measure.nplc = nplc	Set speed (nplc = 0.001 to 25).[2]

1. smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.
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 Series 2600 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 0.5A is set as a rel value on the 1A range, the rel value is also 0.5A on the lower current ranges.

Selecting a range that cannot accommodate the rel value does not cause an overflow condition, but it also does not increase the maximum allowable input for that range. For example, on 1A range, the SourceMeter still overflows for a >1.02A 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[1]	Description
To set rel values:	
smuX.measure.rel.leveli = relval	Set current rel value.
smuX.measure.rel.levelp = relval	Set power rel value.
smuX.measure.rel.levelr = relval	Set resistance rel value.
smuX.measure.rel.levelv = relval	Set voltage rel value.
To enable/disable rel:	
smuX.measure.rel.enablei = smuX.REL_OFF	Disable current rel.
smuX.measure.rel.enablep = smuX.REL_OFF	Disable power rel.
smuX.measure.rel.enabler = smuX.REL_OFF	Disable resistance rel.
smuX.measure.rel.enablev = smuX.REL_OFF	Disable voltage rel.
smuX.measure.rel.enablei = smuX.REL_ON	Enable current rel.
smuX.measure.rel.enablep = smuX.REL_ON	Enable power rel.
smuX.measure.rel.enabler = smuX.REL_ON	Enable resistance rel.
smuX.measure.rel.enablev = smuX.REL_ON	Enable voltage rel.

* smuX = smua for the Model 2601/2611; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612.

Rel programming example

```
smua.reset()
```
--Restore Series 2600 defaults.

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

```
smua.measure.rel.enablei = smua.REL_ON
```
--Enable current rel.

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 three filter types from which to choose. These three filters are broken down into two **averaging filters** and one **median filter**. The median filter is only available on Models 2635 and 2636.

The two averaging filters are 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.

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

When a moving 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. The normal moving filter process continues. Note that a true average or median reading is not yielded until the stack is filled with new reading conversions (no copies in stack). For example, in 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 for Models 2601, 2601, 2611, and 2612

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).

Filter configuration menu for Models 2635 and 2636

Press CONFIG and then FILTER to display the filter configuration menu:

- TYPE - Use this menu item to select filter type (AVERAGE or MEDIAN)
- AVERAGE - Use this menu item to select filter type (MOVING or REPEAT)
- MEDIAN - Use this menu item to select MOVING filter type.
- COUNT - Use this menu item to specify filter count (1 to 100 readings).

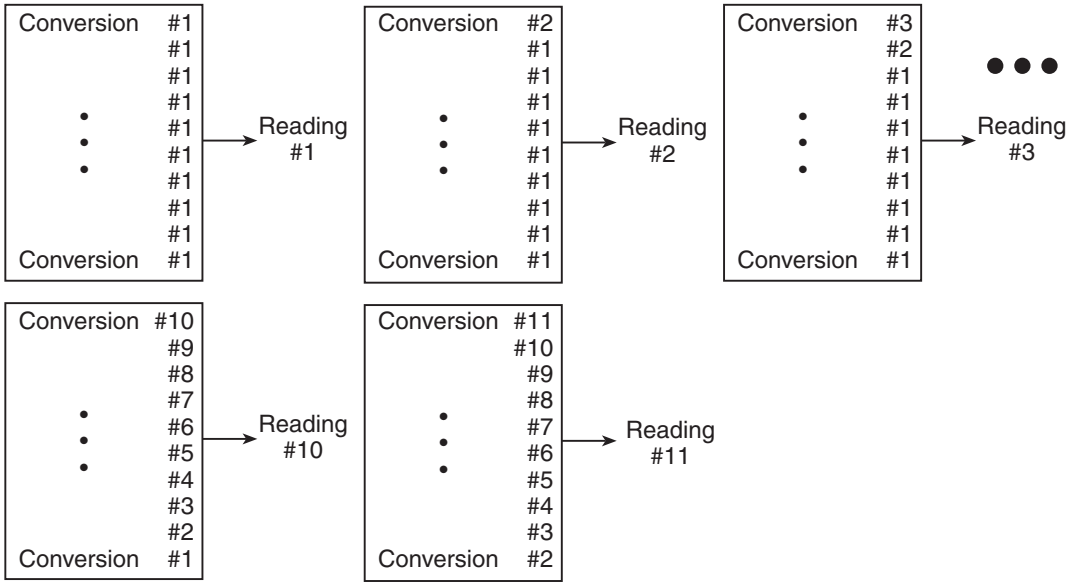
Enabling filter

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.

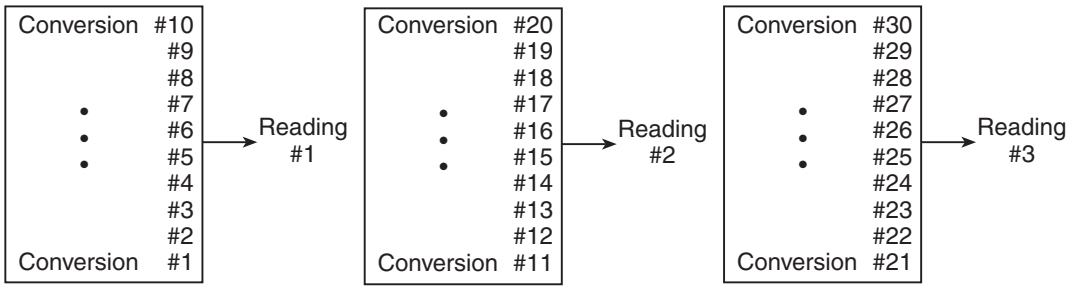
Response time

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

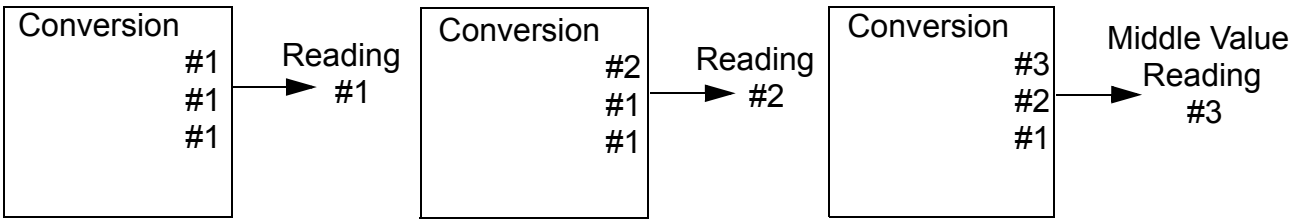


A. Type - Moving Average, Readings = 10



B. Type - Repeating, Readings = 10

Figure 6-2
Median Filter



A. Type - Median, Readings=3

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_MEDIAN	Select median filter type.
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.

* smuX = smua for the Model 2601/2611; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612.

Filter programming example

The example below programs filter aspects as follows:

- Filter type: moving average
- Filter count: 10
- Filter state: enabled

```
smua.reset()
```
--Restore Series 2600 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.

This page left blank intentionally.

Section 7

Buffer (Data Store)

In this section:

Topic	Page
Overview	**7-2**
Data store overview	**7-2**
Front panel data store	**7-2**
Buffer configuration	7-2
Storing readings	7-3
Recalling readings	7-3
Remote data store	**7-4**
Data store commands	7-4
Reading buffers	7-5
Time and date values	7-7
Buffer status	7-8
Dynamically allocated buffers	7-8
Buffer programming examples	7-9

Overview

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

- "Data store overview"
- "Front panel data store"
- "Remote data store"

Data store overview

The Keithley Instruments Series 2600 System 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 time stamp.

Front panel data store

Buffer configuration

The buffer can be configured through the buffer configuration menu, which is accessed as follows:

1. Press **CONFIG** > **STORE**.
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 time stamp).Buffer configuration menu

---

The various buffer configuration menu items include:

- **COUNT:** Sets number of readings to store (1 to 110,000).

NOTE All of the buffers share a common memory area. Therefore, to store as many as 110,000 readings, the source values and time stamps must not be enabled for any of the buffers.

- **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 2602/2612/2636 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 2601/2611/2635 buffer configuration menu items are the same as covered above except for channel selection.

Storing readings

You can append measurements to the reading buffer or override existing measurements in the reading buffer.

Complete the following steps to append or override measurements:

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 **CONFIG** > **STORE** and then select **TRIG-MODE**.
4. Choose one of the following:
   – **APPEND**
   – **OVERRIDE**
5. Press **EXIT** to return to the main menu.

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 2602/2612/2636 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 time stamp (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 2602/2612/2636 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.

Time stamp

If the time stamp is enabled, the first source-measure reading stored in the buffer (#0000001) is time stamped at 0000000.001 seconds. Subsequent readings are time stamped 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 Navigation Wheel to increment and decrement the selected digit of the location number. Cursor position is controlled by the Navigation Wheel 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[1]	Description
smuX.nvbuffer1.clear()	Clear Buffer 1.
smuX.nvbuffer2.clear()	Clear Buffer 2.
mybuffer = smuX.makebuffer(n)	Create dynamically allocated buffer, n readings.
mybuffer = nil	Delete dynamically allocated buffer.
Commands to store readings:[2]	
smuX.measure.count = count	Store count number of buffer readings.
smuX.measure.overlappedi(rbuffer)	Store current readings in buffer.
smuX.measure.overlappediv(ibuffer, vbuffer)	Store current and voltage readings in respective buffers (current and then voltage are stored in separate buffers).
smuX.measure.overlappedp(rbuffer)	Store power readings in buffer.
smuX.measure.overlappedr(rbuffer)	Store resistance readings in buffer.
smuX.measure.overlappedv(rbuffer)	Store voltage readings in buffer.
Commands to access readings:[3]	
print(smuX.measure.i(rbuffer))	Return first buffer current reading.
print(smuX.measure.iv(ibuffer, vbuffer))	Return first current and then voltage reading from two separate buffers.
print(smuX.measure.p(rbuffer))	Return first buffer power reading.
print(smuX.measure.r(fbuffer))	Return first buffer resistance reading.
print(smuX.measure.v(rbuffer))	Return first buffer voltage reading.
printbuffer(start_index, end_index, st_1 [, st_n])	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). [4]
printnumber(v1[, vn])	Print numbers with selected buffer format: v1 … vn (Numbers to print).

1. smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.
2. rbuffer, ibuffer, and vbuffer = smuX.nvbuffer1 or smuX.nvbuffer2.
3. rbuffer, ibuffer, and vbuffer = smuX.nvbuffer1 or smuX.nvbuffer2
4. See "Reading buffers" below 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, while read-only attribute programming examples are listed in Table 7-5.

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 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]. This attribute is initialized to off when the buffer is created.
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 or on, can only be changed when the buffer is empty. When the buffer is created, this attribute is initialized to off.
collecttimestamps	When this attribute is on, time stamps will be stored with readings in the buffer. This requires 4 extra bytes of storage per reading. This value, off or on, can only be changed when the buffer is empty. When the buffer is created, this attribute is initialized to off.
timestampresolution	The time stamp resolution, in seconds. When the buffer is created, its initial resolution is 0.000001 seconds. At this resolution, the reading buffer can store unique time stamps 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 time stamp of when the reading at rb[1] was stored, in seconds from power-up.
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
smua.nvbuffer1.collectsourcevalues = 1	Enable source value storage.
smua.nvbuffer1.appendmode = 1	Enable buffer append mode.
smua.nvbuffer1.collecttime stamps = 0	Disable time stamp storage.
smua.nvbuffer1.timestampresolution = 0.001	Set time stamp resolution to 0.001s.

Table 7-5
Buffer read-only attribute programming examples

Command	Description
number = smua.nvbuffer1.n	Request number of readings in buffer.
buffer_size = smua.nvbuffer1.capacity	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[1]	Description
`measurefunctions`	An array (a LUA table) of strings indicating the function measured for the reading (Current, Voltage, Ohms or Watts).
`measureranges`	An array (a LUA table) of full scale range values for the measure range used when the measurement was made.
`readings`	An array (a LUA table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly, i.e., rb[2] and rb.readings[2] are the same value.
`sourcefunctions`	An array (a LUA table) of strings indicating the source function at the time of the measurement (Current or Voltage).
`sourceoutputstates`	An array (a LUA table) of strings indicating the state of the source (Off or On).
`sourceranges`	An array (a LUA table) of full scale range values for the source range used when the measurement was made.
`sourcevalues`	If enabled, an array (a LUA table) of the sourced values in effect at the time of the reading.
`statuses`	An array (a LUA table) of status values for all of the readings in the buffer. The status values are floating-point numbers that encode the status value into a floating-point value (see Table 7-7).
`timestamps`	An array (a LUA table) of time stamps, in seconds, of when each reading occurred. These are relative to the `basetime stamp` for the buffer (Table 7-3).

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 earlier in this section in “"Reading buffer designations".”
.

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 Series 2600 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()                                    - Restore Series 2600 defaults.
                                                -
smua.measure.autorangei = smua.AUTORANGE_ON     - Select measure I auto range.
                                                -
smua.measure.autorangev = smua.AUTORANGE_ON     - Select measure V auto range.
                                                -
format.data = format.ASCII                      - Select ASCII data format.
                                                -
smua.nvbuffer1.clear()                          - Clear buffer 1.
                                                -
smua.nvbuffer2.clear()                          - Clear buffer 2.
                                                -
smua.measure.count = 100                        - Set buffer count to 100.
                                                -
smua.measure.interval = 0.1                     - Set measure interval to 0.1s.
                                                -
smua.source.func = smua.OUTPUT_DCVOLTS          - Select source voltage function.
                                                -
smua.source.levelv = 1                          - Output 1V.
                                                -
smua.source.output =smua.OUTPUT_ON              - Turn on output.
                                                -
smua.measure.overlappediv                       - Store current readings in buffer 1,
(smua.nvbuffer1,smua.nvbuffer2)                   voltage readings in buffer 2.

waitcomplete()                                  - Wait for buffer to fill.
                                                -
smua.source.output =smua.OUTPUT_OFF             - Turn off output.
                                                -
printbuffer(1, 100, smua.nvbuffer1)             - Return buffer 1 readings 1-100.
                                                -
printbuffer(1, 100, smua.nvbuffer2)             - 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()                                    - Restore Series 2600 defaults.
                                                -
smua.measure.autorangei = smua.AUTORANGE_ON     - Select measure auto range.
                                                -
format.data = format.ASCII                      - Select ASCII data format.
                                                -
mybuffer = smua.makebuffer(100)                 - Allocate mybuffer, 100 readings.
                                                -
smua.measure.count = 100                        - Set buffer count to 100.
                                                -
smua.measure.interval = 0.1                     - Set measure interval to 0.1s.
                                                -
smua.source.func = smua.OUTPUT_DCVOLTS          - Select source voltage function.
                                                -
smua.source.levelv = 1                          - Output 1V.
                                                -
smua.source.output =smua.OUTPUT_ON              - Turn on output.
                                                -
smua.measure.overlappedi(mybuffer)              - Store current readings in mybuffer.
                                                -
waitcomplete()                                  - Wait for buffer to fill.
                                                -
smua.source.output =smua.OUTPUT_OFF             - Turn off output.
                                                -
printbuffer(1, 100, mybuffer)                   - Return readings 1-100 from
                                                - mybuffer.
mybuffer = nil                                  - Delete mybuffer.
                                                -
```

This page left blank intentionally.

Section 8

Source-Measure Concepts

In this section:

Topic	Page
Overview	**8-2**
Compliance limit	**8-2**
Maximum compliance	8-2
Compliance principles	8-3
Sweep waveforms	**8-3**
Staircase sweeps	8-3
Pulse sweeps	8-4
Overheating protection	**8-4**
Power equations to avoid overheating	8-4
Operating boundaries	**8-7**
Source or sink	8-7
Continuous power operating boundaries	8-8
I-Source operating boundaries	8-9
V-Source operating boundaries	8-13
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)	8-17
Contact check	8-18
Guard	**8-19**
Guard overview	8-19
Guard connections	8-20
Pulse concepts	**8-22**
Pulse period	8-22
Pulse rise and fall times	8-22
Pulse duty cycle	8-23
Settling time considerations	**8-23**
Measurement Settling Time Considerations	8-23
Reduction in gain-bandwidth	8-24

Overview

The documentation in this section provides detailed information on source-measure concepts and includes the following information:

- "Compliance limit"
- "Sweep waveforms"
- "Overheating protection"
- "Operating boundaries"
- "Basic circuit configurations"
- "Guard"
- "Pulse concepts"

Compliance limit

When sourcing voltage, the Keithley Instruments Series 2600 System 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, except for the condition described in "Compliance limit" in Section 4.

Maximum compliance

The maximum compliance values for the source ranges are summarized in Table 8-1.

Table 8-1
Maximum compliance limits

Model 2601/2602		**Model 2611/2612**		**Model 2635/2636**	
Source range	**Maximum compliance value**	**Source range**	**Maximum compliance value**	**Source range**	**Maximum compliance value**
100mV	3A	200mV	1.5A	200mV	
1V	3A	2V	1.5A	2V	
6V	3A	20V	1.5A	20V	
40V	1A	200V	100mA	200V[1]	
	40V	100nA	200V	100pA	
	40V	1µA	200V	1nA	
	40V	10µA	200V	10nA	
	40V	100µA	200V	100nA	
	40V	1mA	200V	1µA	
	40V	10mA	200V	10µA	
	40V	100mA	200V	100µA	
	40V	1A	20V	1mA	
	6V	1.5A	20V	10mA	
				100mA	
				1A	
				1.5A	

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: 10Ω

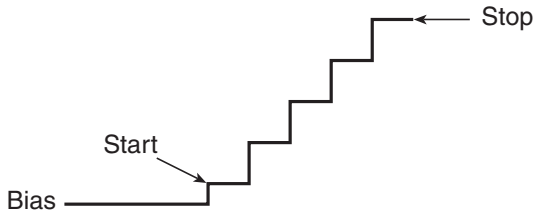
With a source voltage of 10V and a DUT resistance of 10Ω, the current through the DUT should be: 10V/10Ω = 1A. However, because the compliance is set to 10mA, the current will not exceed that value, and the voltage across the resistance is limited to 100mV. In effect, the 10V voltage source is transformed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

Sweep waveforms

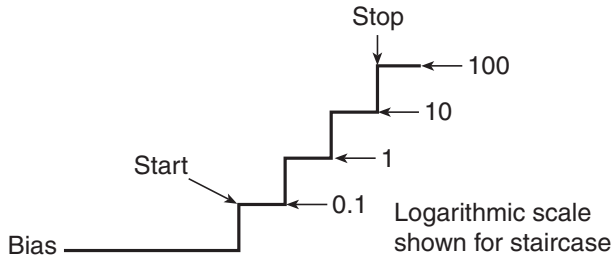
Staircase sweeps

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 equilinear 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 staircase sweep waveforms



A: Linear Staircase Sweep



B: Logarithmic Staircase Sweep

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

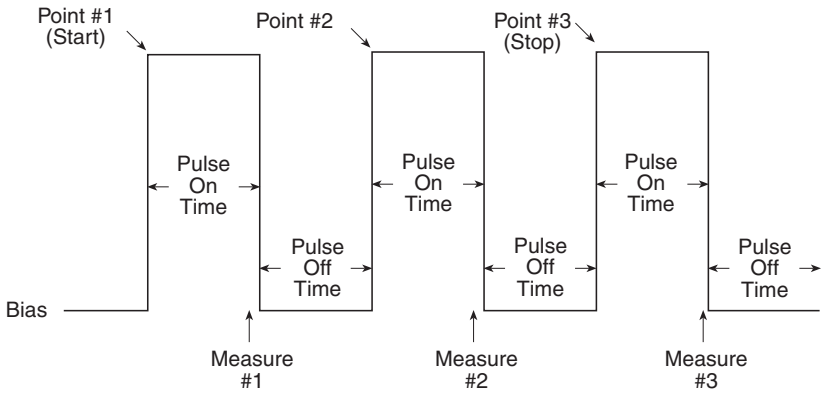
The Series 2600 can also perform the following pulse sweeps:

- 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" in Section 5 and "Pulse concepts" described later in this section 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. Even with proper ventilation, the SourceMeter can overheat if the ambient temperature is too high or the SourceMeter is being operated in sink mode for long periods of time. 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

To avoid overheating, each SourceMeter channel should not be operated in a manner that would force it to exceed the maximum duty cycle (DC_{MAX}) computed using the "General SourceMeter power equation" below. Factors such as the ambient temperature, quadrant of operation, and high power pulse levels (if applicable) affect the maximum duty cycle. Exceeding the calculated maximum duty cycle may cause the SourceMeter's over temperature protection mechanism to engage. When this happens, an error message will be displayed, and the SourceMeter output(s) will be disabled until the internal temperature of the SourceMeter is reduced to an acceptable level.

You do **not** have to be concerned about overheating if **all** of the following are true:

- The SourceMeter is being used as a power source and not a power sink.
- The ambient temperature is ≤ 30°C.

- High power pulse operation is not being used.

However, if any one of these is false, the SourceMeter may overheat if operated in a manner that exceeds the calculated maximum duty cycle, DC_{MAX}.

The maximum duty cycle equation is derived from the power equation below by solving for DC_{MAX}. The general power equation describes how much power a SourceMeter channel can source and/or sink before the total power cannot be fully dissipated by the SourceMeter cooling system. This equation takes into account all of the factors that can influence the power being dissipated by the SourceMeter.

General SourceMeter power equation

$$\left|(V_{OA} - V_P)(I_P)\right|\sqrt{DC_{MAX}} + \left|(V_{OA} - V_B)(I_B)\right| \leq (P_{CS} - T_{DER})$$

P_{CS} = The maximum power generated in a SourceMeter channel that can be properly dissipated by the SourceMeter cooling system.

T_{AMB} = The ambient temperature of the SourceMeter operating environment.

$T_{DER} = T_{AMB} - 30$

- This factor represents the number of watts the SourceMeter is de-rated when operating in environments above 30°C. This is represented as a temperature because the maximum output power of each SourceMeter channel is reduced by 1W per degree C above 30°C.
- T_{DER} is 0 when the ambient temperature is below 30°C.

V_{OA} = The SourceMeter output amplifier voltage. This constant can be found in the tables below.

V_P = The voltage level the SourceMeter is attempting to force while at the pulse level.

- When operating in quadrants 1 or 3 (sourcing power), the sign of this voltage must be **positive** when used in the power equations.
- When operating in quadrants 2 or 4 (sinking power), the sign of this voltage must be **negative** when used in the power equations.

V_B = The voltage level the SourceMeter is attempting to force while at the bias level.

- When operating in quadrants 1 or 3 (sourcing power), the sign of this voltage must be **positive** when used in the power equations.
- When operating in quadrants 2 or 4 (sinking power), the sign of this voltage must be **negative** when used in the power equations.

I_P = The current flowing through the SourceMeter channel while at the pulse level.

I_B = The current flowing through the SourceMeter channel while at the bias level.

Maximum duty cycle equation [1]

$$DC_{MAX} \leq \left[\frac{(P_{CS} - T_{DER}) - \left|(V_{OA} - V_B)(V_B)\right|}{\left|(V_{OA} - V_P)(I_P)\right|} \right]^2 \times 100$$

NOTE When attempting to determine the maximum duty cycle where the off state will be 0V or 0A:

- I_B is 0
- I_P and V_P are the voltage and current levels when the SourceMeter is on.

CAUTION This maximum duty cycle equation is an approximation. In general, if the duty cycle calculation yields a number > 90%, then DC under those conditions should not cause the SourceMeter to overheat. However, if the calculation yields a number < 10%, the calculated duty cycle should not be exceeded by more than 0.5% to avoid potential overheating.

Table 8-2
Model 2601/2602 Maximum Duty Cycle equation constants

Constant	100mV range	1V range	6V range
P_{HS}	56	56	56
V_{OA}	18	18	18

Table 8-3
Model 2611/2612/2635/2636 Maximum Duty Cycle equation constants

Constant	100mV range	1V range	6V range
P_{HS}	56	56	56
V_{OA}	18	18	18

Examples

Example 1:

Using a Model 2611 to charge a 5V battery with 1.5A, while operating at 50°C ambient temperature; what is the maximum duty cycle?

1. Equations apply to both channels, sinking or sourcing power simultaneously. If a duty cycle less than 100% is required to avoid overheating, the maximum on time must be less than 10 seconds

Assuming the 20V range will be used to measure the voltage:

$$DC_{MAX} \leq \left[\frac{(56-20)-|(38-(5))(0)|}{|(38-(5))(1.5)|}\right]^2 \times 100$$

$$DC_{MAX} \leq 52.9\%$$

Example 2:

Using a Model 2602 to pulse 10A of current from a bias level of 500mA, into a very low impedance (100 mΩ), while operating at 40°C ambient temperature; what is the maximum duty cycle?

Assuming the 1V range will be used to measure the voltage:

$$DC_{MAX} \leq \left[\frac{(56-10)-|(18-(0.1)(0.5))(0.5)|}{|(18-(10)(0.1))(10)|}\right]^2 \times 100$$

$$DC_{MAX} \leq 4.7\%$$

Example 3:

Using a Model 2612 to charge a 12V battery by sourcing 100mA and then discharging the battery by sinking 5A, while operating at 35°C ambient temperature; what is the maximum duty cycle?

Assuming the 20V range will be used to measure the voltage:

$$DC_{MAX} \leq \left[\frac{(56-5)-|(38-(12))(0.1)|}{|(38-(-12))(-5)|}\right]^2 \times 100$$

$$DC_{MAX} \leq 3.7\%$$

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 and Figure 8-4. 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.

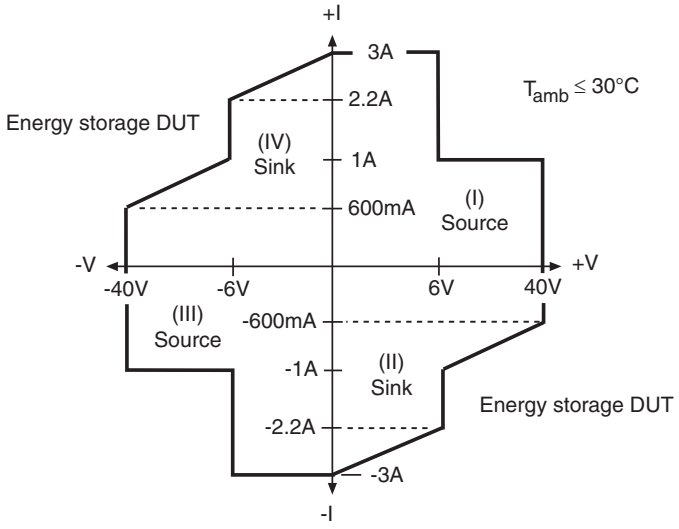
When operating in the second (II) or fourth (IV) quadrant, the SourceMeter is operating as a sink (V and I have opposite polarity). As a sink, it is dissipating power rather than sourcing it. An external source or an energy storage device, such as a capacitor or battery, can force operation in the sink region.

Continuous power operating boundaries

Model 2601/2602 continuous power operating boundaries

The general operating boundaries for Model 2601/2602 continuous power output are shown in Figure 8-3 (for derating factors, see "General SourceMeter power equation", described earlier in this section). In this drawing, the 3A, 6V and 1A, 40V magnitudes are nominal values. Also note that the boundaries are not drawn to scale.

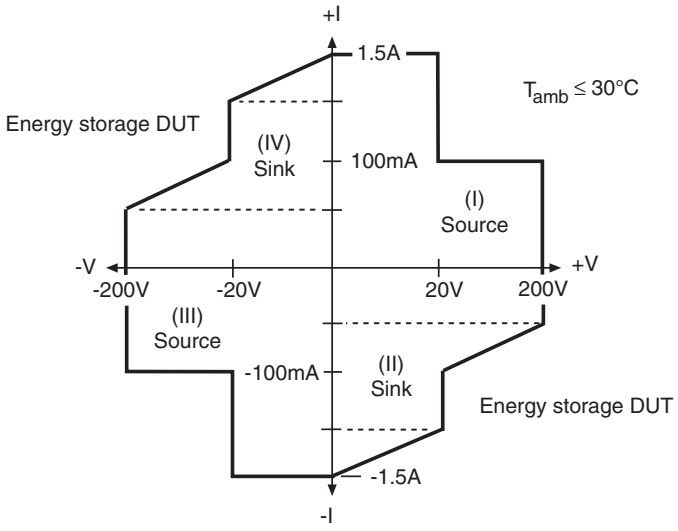
Figure 8-3
Model 2601/2602 continuous power operating boundaries



Model 2611/2612 continuous power operating boundaries

The general operating boundaries for Model 2611/2612 continuous power output are shown in Figure 8-4 (see "General SourceMeter power equation" in this section for derating factors). In this drawing, the 1.5A, 20V and 100mA, 200V magnitudes are nominal values. Also note that the boundaries are not drawn to scale.

Figure 8-4
Model 2611/2612/2635/2636 continuous power operating boundaries



I-Source operating boundaries

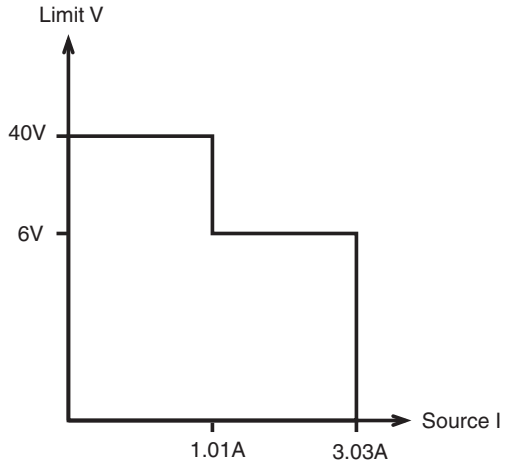
Model 2601/2602 I-Source operating boundaries

Figure 8-5 shows the operating boundaries for the I-Source. Only the first quadrant of operation is covered. Operation in the other three quadrants is similar.

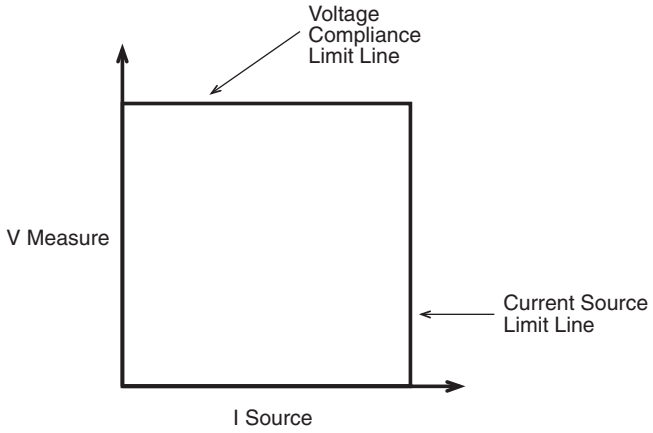
Figure 8-5A shows the output characteristics for the I-Source. As shown, the Model 2601/2602 SourceMeter can output up to 1.01A at 40V, or 3.03A at 6V. Note that when sourcing more than 1.01A, voltage is limited to 6V.

Figure 8-5B shows the limit lines for the I-Source. The current source limit line represents the maximum source value possible for the presently selected current source range. The voltage compliance limit line represents the actual compliance that is in effect. See "Compliance limit" earlier in this section. 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 8-5
Model 2601/2602 I-Source boundaries



A) Output Characteristics



B) Limit Lines

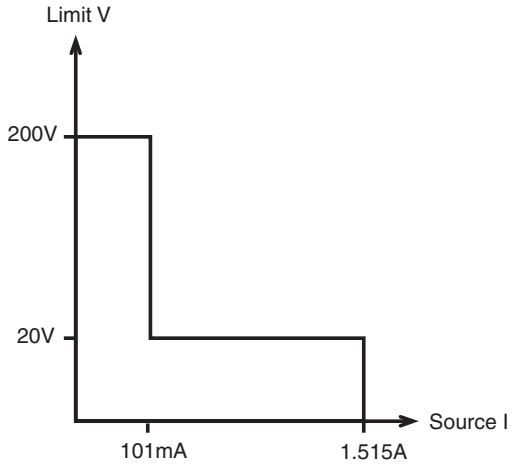
Model 2611/2612/2635/2636 I-Source operating boundaries

Figure 8-6 shows the operating boundaries for the I-Source. Only the first quadrant of operation is covered. Operation in the other three quadrants is similar.

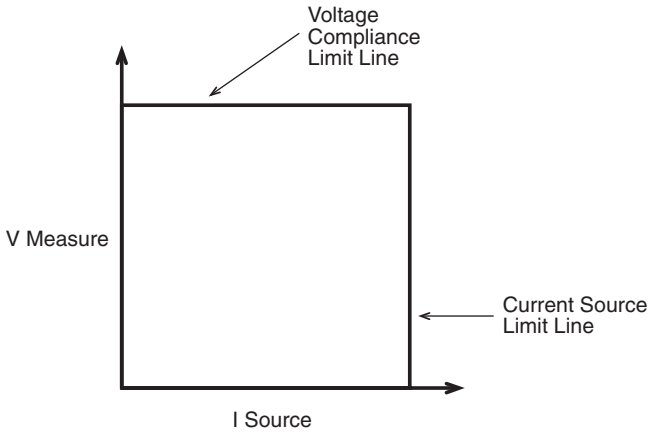
Figure 8-6A shows the output characteristics for the I-Source. As shown, the Model 2611/2612/2635/2636 SourceMeter can output up to 101mA at 200V, or 1.515A at 20V. Note that when sourcing more than 101mA, voltage is limited to 20V.

Figure 8-6B shows the limit lines for the I-Source. The current source limit line represents the maximum source value possible for the presently selected current source range. The voltage compliance limit line represents the actual compliance that is in effect. See “"Compliance limit"” earlier in this section. 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
Model 2611/2612/2635/2636 I-Source boundaries



A) Output Characteristics



B) Limit Lines

Load considerations

The boundaries the SourceMeter operates in depends on the load (DUT) that is connected to its output. Figure 8-7 shows operation examples for resistive loads that are 50Ω and 200Ω, respectively. For these examples, the SourceMeter is programmed to source 100mA and limit 10V.

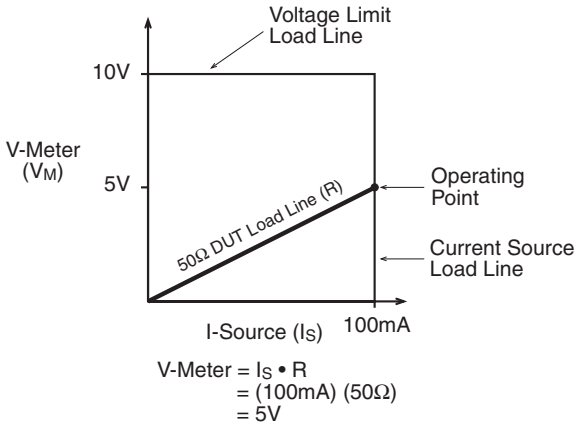
In Figure 8-7A, the SourceMeter is sourcing 100mA to the 50Ω load and subsequently measures 5V. As shown, the load line for 50Ω intersects the 100mA current source line at 5V.

Figure 8-7B shows what happens if the resistance of the load is increased to 200Ω. The DUT load line for 200Ω intersects the voltage compliance limit line placing the SourceMeter in compliance. In compliance, the SourceMeter will not be able to source its programmed current (100mA). For the 200Ω DUT, the SourceMeter will only output 50mA (at the 10V limit).

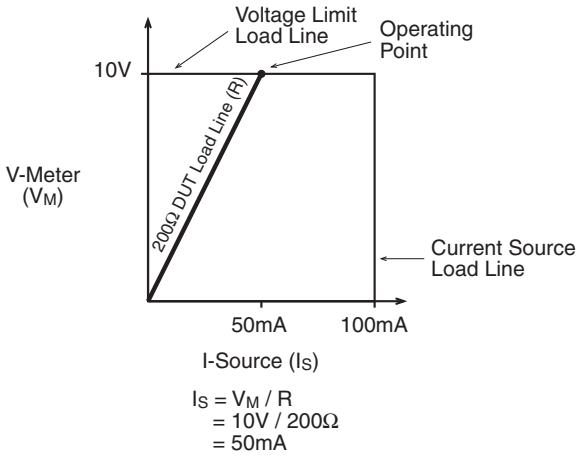
Notice that as resistance increases, the slope of the DUT load line increases. As resistance approaches infinity (open output), the SourceMeter will source virtually 0mA at 10V. Conversely, as resistance decreases, the slope of the DUT load line decreases. At zero resistance (shorted output), the SourceMeter will source 100mA at virtually 0V.

Regardless of the load, voltage will never exceed the programmed compliance of 10V.

Figure 8-7
I-Source operating examples



A) Normal I-source operation



B) I-source in compliance

V-Source operating boundaries

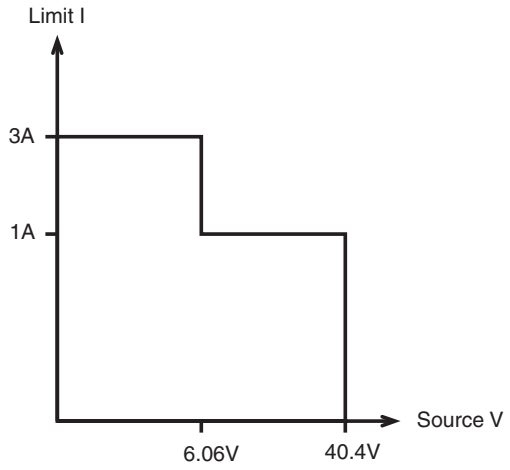
Model 2601/2602 V-Source operating boundaries

Figure 8-8 shows the operating boundaries for the V-Source. Only the first quadrant of operation is covered. Operation in the other three quadrants is similar.

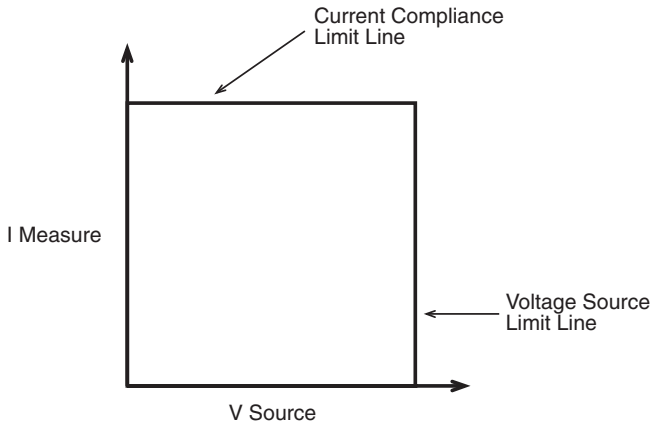
Figure 8-8A shows the output characteristics for the V-Source. As shown, the Model 2601/2602 SourceMeter can output up to 6.06V at 3A, or 40.4V at 1A. Note that when sourcing more than 6.06V, current is limited to 1A.

Figure 8-8B shows the limit lines for the V-Source. The voltage source limit line represents the maximum source value possible for the presently selected voltage source range. For example, if you are using the 6V source range, the voltage source limit line is at 6.3V. The current compliance limit line represents the actual compliance in effect. See “"Compliance limit"” earlier in this section. 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 8-8
Model 2601/2602 V-Source boundaries



A) Output characteristics



B) Limit lines

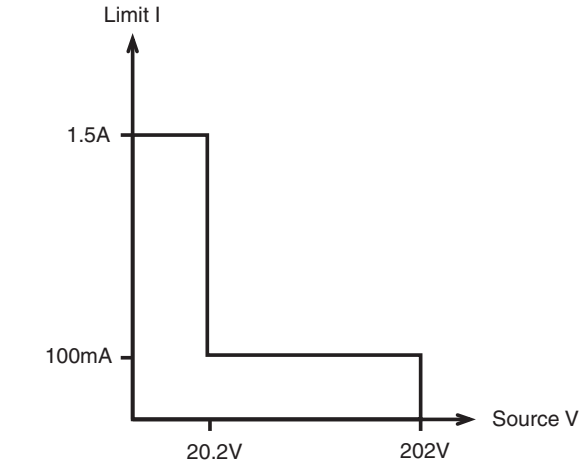
Model 2611/2612/2635/2636 V-Source operating boundaries

Figure 8-9 shows the operating boundaries for the V-Source. Only the first quadrant of operation is covered. Operation in the other three quadrants is similar.

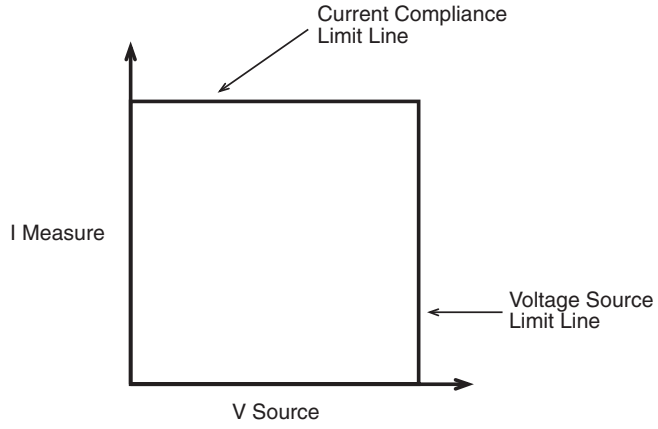
Figure 8-9A shows the output characteristics for the V-Source. As shown, the Model 2611/2612/2635/2636 SourceMeter can output up to 20.2V at 1.5A, or 202V at 100mA. Note that when sourcing more than 20.2V, current is limited to 100mA.

Figure 8-9B shows the limit lines for the V-Source. The voltage source limit line represents the maximum source value possible for the presently selected voltage source range. For example, if you are using the 20V source range, the voltage source limit line is at 20.2V. The current compliance limit line represents the actual compliance in effect. See "'Compliance limit'" earlier in this section. 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 8-9
Model 2611/2612/2635/2636 V-Source boundaries



A) Output characteristics



B) Limit lines

Load considerations

The boundaries the SourceMeter operates in depends on the load (DUT) that is connected to the output. Figure 8-10 shows operation examples for resistive loads that are 2kΩ and 800Ω, respectively. For these examples, the SourceMeter is programmed to source 10V and limit 10mA.

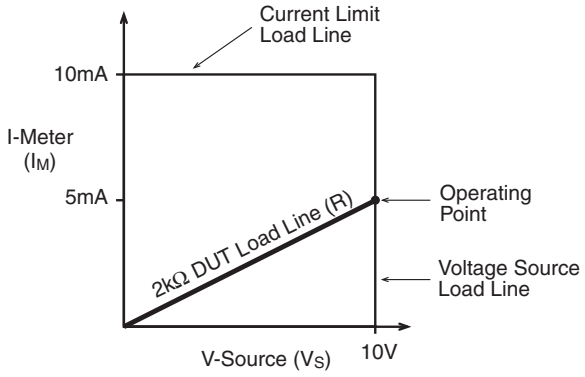
In Figure 8-10A, the SourceMeter is sourcing 10V to the 2kΩ load and subsequently measures 5mA. As shown, the load line for 2kΩ intersects the 10V voltage source line at 5mA.

Figure 8-10B shows what happens if the resistance of the load is decreased to 800Ω. The DUT load line for 800Ω intersects the current compliance limit line placing the SourceMeter in compliance. In compliance, the SourceMeter will not be able to source its programmed voltage (10V). For the 800Ω DUT, the SourceMeter will only output 8V (at the 10mA 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 10V at 0mA. Conversely, as resistance increases, the slope of the DUT load line decreases. At zero resistance (shorted output), the SourceMeter will source virtually 0V at 10mA.

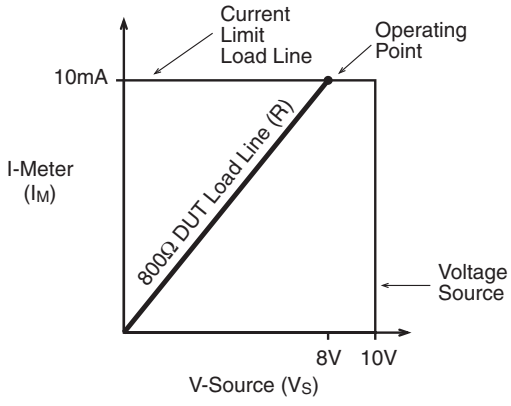
Regardless of the load, current will never exceed the programmed compliance of 10mA.

Figure 8-10
V-Source operating examples



$I_M = V_S / R$
$= 10V/2k\Omega$
$= 5mA$

A) Normal V-source operation



$V_S = I_M \bullet R$
$= (10mA)\ (800\Omega)$
$= 8V$

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

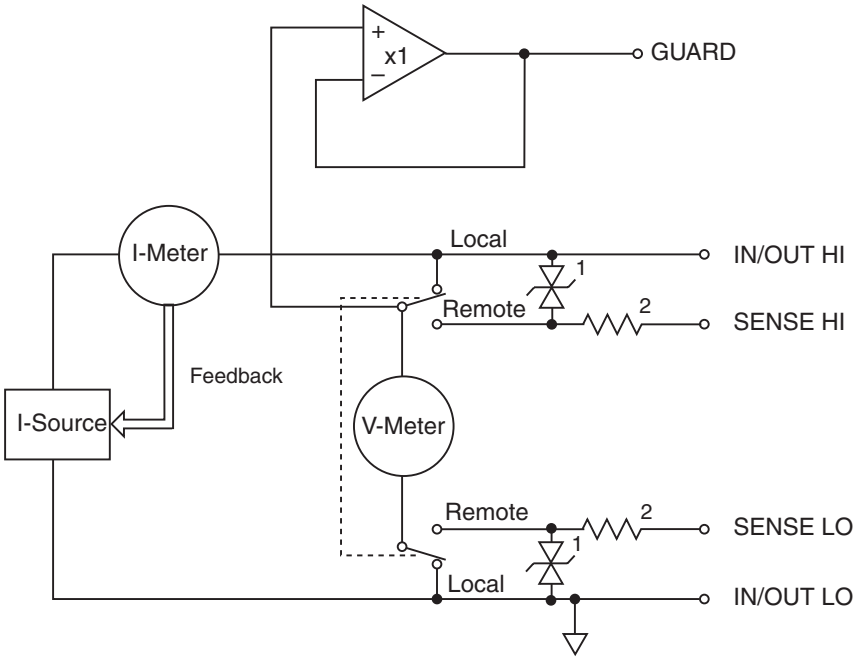
Source I

When configured to source current (I-Source) as shown in Figure 8-11, 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 8-11
Source I configuration



NOTES: 1. This represents a protection circuit that is very high impedance until the voltage across it exceeds approximately 3V. Above 3V, the protection turns on and allows current to flow through it.
2. Approximately 13kΩ.

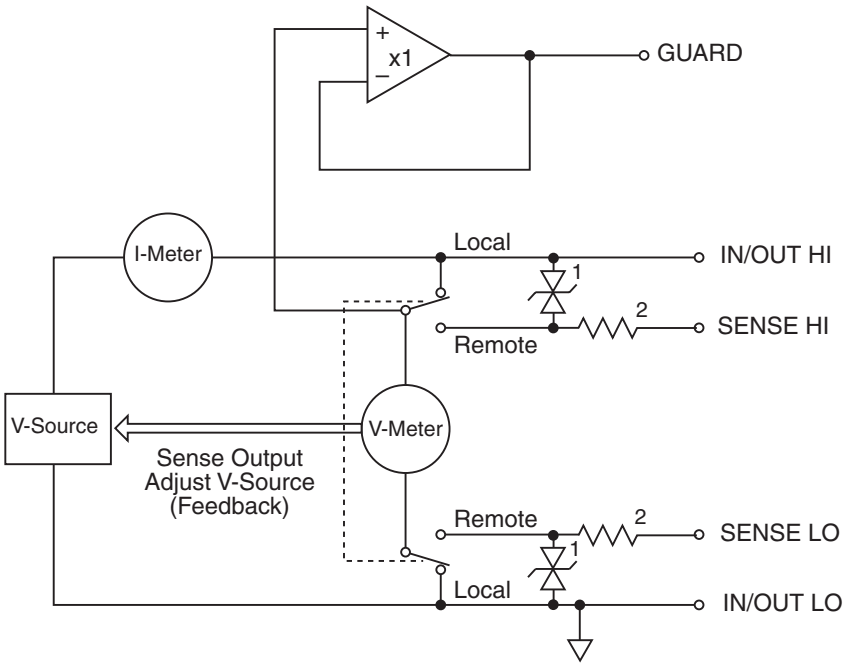
Source V

When configured to source voltage (V-Source) as shown in Figure 8-12, 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 8-12
Source V configuration



NOTES: 1. This represents a protection circuit that is very high impedance until the voltage across it exceeds approximately 3V. Above 3V, the protection turns on and allows current to flow through it.
2. Approximately 13kΩ.

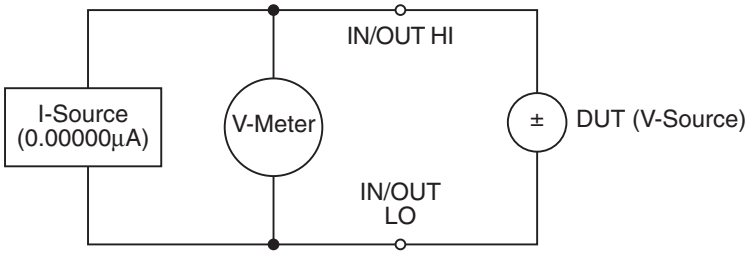
Measure only (V or I)

Figure 8-13 shows the configurations for using the SourceMeter exclusively as a voltmeter or ammeter. As shown in Figure 8-13A, the SourceMeter is configured to measure voltage-only by setting it to source 0A and measure voltage.

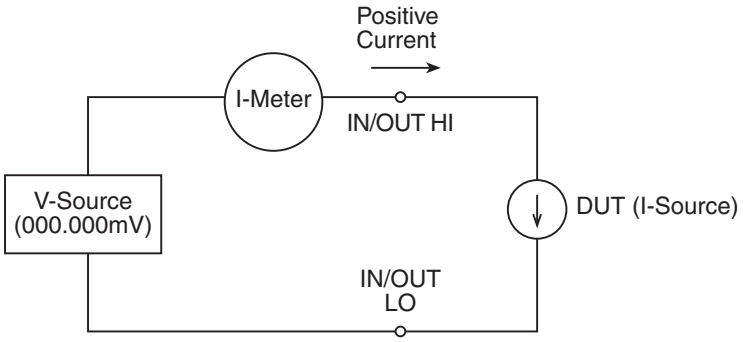
CAUTION V-Compliance must be set to a level that is higher than the measured voltage. Otherwise, excessive current will flow into the SourceMeter. This current could damage the SourceMeter. Also, when connecting an external voltage to the I-Source, set the output off state to the high-impedance mode. See "'Compliance limit'" earlier in this section for details.

In Figure 8-13B, 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 8-13
Measure only configurations



A. Measure Voltage Only



Note: Positive current flowing out of IN/OUT HI results in positive (+) measurements.

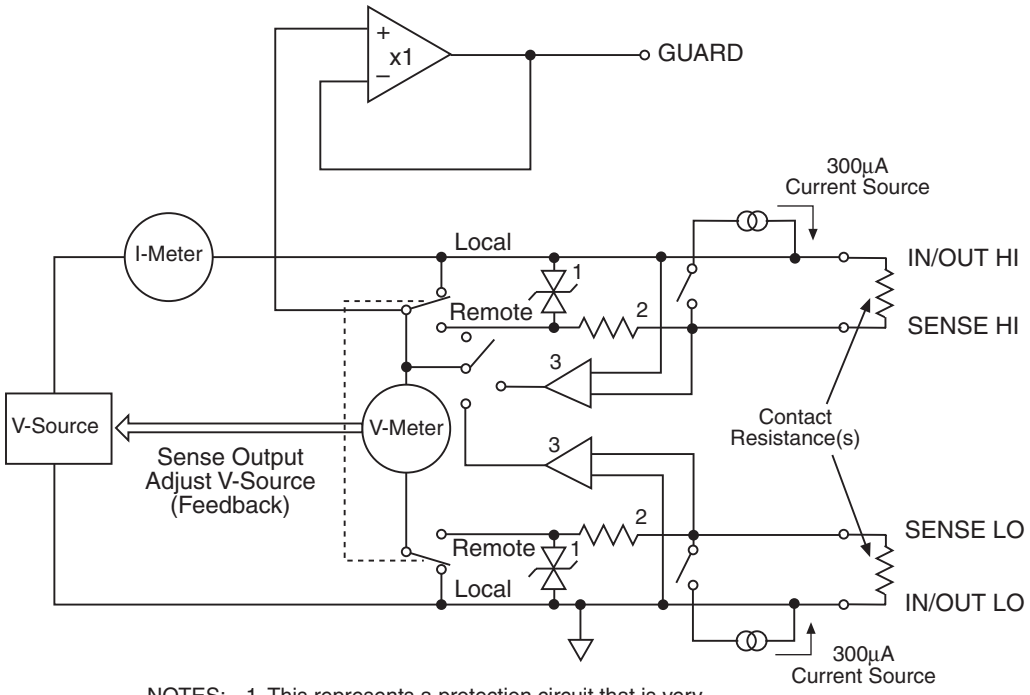
B. Measure Current Only

Note: Use 2-wire local sensing.

Contact check

When a contact check measurement is being performed, two small current sources are switched in between the HI and SENSE HI terminals and the LO and SENSE LO terminals. By controlling the switches illustrated in Figure 8-14, the current from these sources flows through the test leads and through the contact resistance as shown. To accurately measure the resulting contact resistance, the differential amplifier outputs are measured once with the current sources connected, and again with the current sources disconnected. This allows for compensation of various offset voltages that can occur.

Figure 8-14
Contact check circuit configuration



NOTES:
1. This represents a protection circuit that is very high impedance until the voltage across it exceeds approximately 3V. Above 3V, the protection turns on and allows current to flow through it.
2. Approximately 13kΩ.
3. High impedance differential amplifier.

Guard

WARNING ***GUARD is at the same potential as output HI. Thus, if hazardous voltages are present at output HI, they are also present at the GUARD terminal.***

Guard overview

The driven guard (available at the rear panel GUARD terminals) is always enabled and provides a buffered voltage that is at the same level as the Input/Output HI (or Sense HI for remote sense) voltage. The purpose of guarding is to eliminate the effects of leakage current (and capacitance) that can exist between input/output high and low. In the absence of a driven guard, leakage in the external test circuit could be high enough to adversely affect the performance of the 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 8-15B shows the metal case of a test fixture being used as a safety shield.***

NOTE See Section 3 for details on guarded test connections.

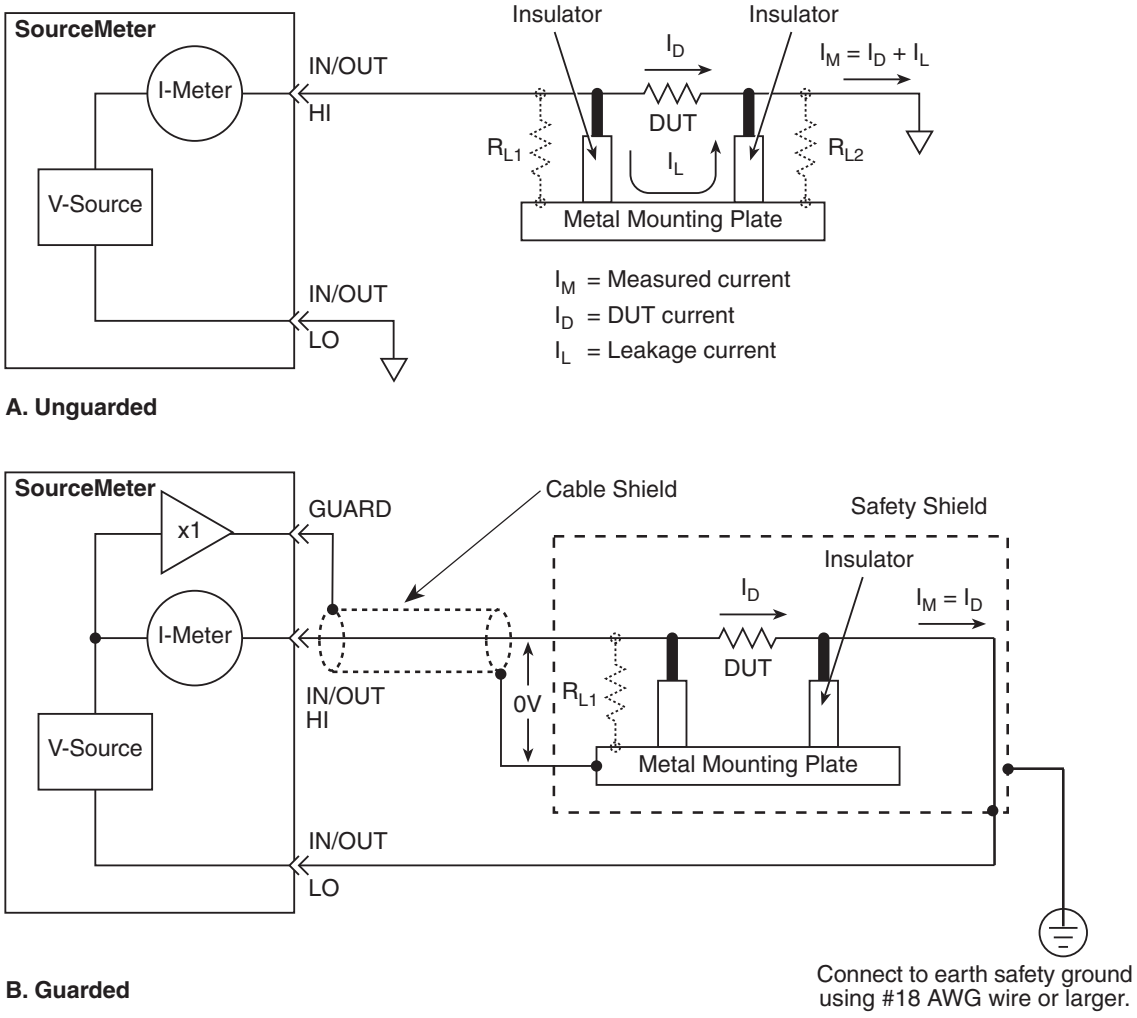
Inside the test fixture, a triaxial cable can be used to extend guard to the DUT. The center conductor of the cable is used for In/Out HI, the inner shield is used for guard, and the outer shield is used for In/Out LO and is connected to the safety shield (which is connected to safety earth ground).

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 8-15 shows how cable guard can eliminate leakage current through the insulators in a test fixture. In Figure 8-15A, 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 8-15B, 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 8-15
Comparison of unguarded and guarded measurements



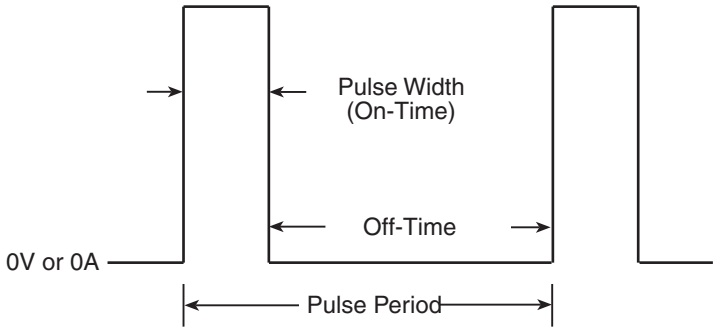
Pulse concepts

Using factory scripts, the Series 2600 can perform fixed, linear staircase, and logarithmic staircase pulse sweeps (see “"Pulse sweeps"” in Section 5 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. Additional settling time considerations are discussed in “"Settling time considerations"” later in this section.

Pulse period

As shown in Figure 8-16, 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.

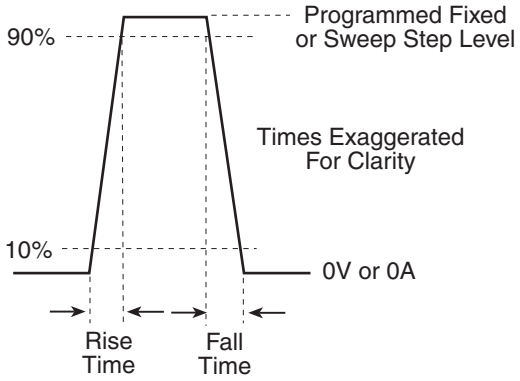
Figure 8-16
Pulse period



Pulse rise and fall times

As shown in Figure 8-17, 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 Series 2600, 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 8 for details on transient response and source settling times.

Figure 8-17
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:

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

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

Duty cycle = 10msec / (10msec + 90msec)
= 10msec / 100msec
= 0.10
= 10%

Settling time considerations

Measurement Settling Time Considerations

Several outside factors can influence measurement settling times. Effects such as dielectric absorption, cable leakages, and noise can all extend the times required to make stable measurements. Care should be taken to use appropriate shielding, guarding, and aperture selections when making low current measurements.

Each current measurement range has a combination of a range resistor and a compensating capacitor that must settle out to allow a stable measurement. By default (on power up or after smuX.reset()), delays are enforced to account for approximately 7τ or 7 time constants of a given range (to reach 0.1% of the final value, assuming 2.3τ per decade). The table below lists the current ranges and associated default delays. In addition, a 1Hz analog filter is used by default on the 1nA and 100pA ranges.

Table 8-4
Current Measure Settling Time[1, 2]

Time required to reach 0.1% of final value after source level command is processed on a fixed range. • Values below for Vout = 2V unless noted	
Current range	**Settling time**
1.5A to 1A	<120µs (typical)(Rload>6Ω)
100mA to 10mA	<80µs (typical)
1mA	<100µs (typical)
100µA	<150µs (typical)
10µA	<500µs (typical)
1µA	<2.5ms (typical)
100nA	<15ms (typical)
10nA	<90ms (typical)
1nA[1]	<360ms (typical)
100pA[3]	<360ms (typical)

1. Delay factor set to 1. Compliance equal to 100 mA.
2. Time for measurement to settle after a Vstep.
3. With default analog filter setting < 450ms.

Both the analog filter and the default delays can be manipulated for faster response times. The analog filter may be turned off to yield faster settling times. The default delays may also be controlled by using the delay factor multiplier. The default value for delay factor multiplier is 1.0, but adjusting to other values will result in either a faster or slower response. For example,

increasing the delay factor to 1.3 will account for settling to 0.01% of the final value. The commands to manipulate the delay factor and analog filter are shown below:

For controlling settling time

```
smuX.measure.delay = 0
```
-- to turn off measure delay (default setting is smuX.DELAY_AUTO)

```
smuX.measure.delay = Y
```
-- set measure delay for all ranges to Y (in seconds)

```
smuX.measure.delayfactor = 1.0
```
--To adjust the delays factor:

This factor is used to multiply the default delays. Setting this value above 1.0 increases the delays, while a value below 1.0 decreases the delay. Setting this value to 0.0 essentially turns off measurement delays. This attribute is only used when:

```
smuX.measure.delay = smuX.DELAY_AUTO.
```

For analog filter

```
smuX.measure.analogfilter = 1
```
(default)

This filter is only active when the amps measure range is 1na/100pA. Setting the attribute to zero disables the filter.

Reduction in gain-bandwidth

The settling time of the SMU can be influenced by the impedance of the DUT in several ways. One influence is caused by an interaction between the impedances of the SMU current source feedback element and the DUT. This interaction can cause a reduction in gain-bandwidth. When the SMU gain-bandwidth is reduced, the settling time of the current source increases.

The settling times listed in the specifications found in Appendix A assume that the gain-bandwidth of the SMU is approximately 60kHz. Table 8-5 below can be used to determine the affect of various DUT impedances on the gain-bandwidth when the SMU is operating on each current source range. If the ratio of DUT impedance to current source feedback impedance drops below the indicated 60kHz ratio, then the settling time will increase beyond the specified times. Therefore, there is a minimum DUT impedance for each current source range. The settling time on a current range can increase significantly when measuring DUTs that have an impedance that is lower than that listed in Table 8-5 below.

Table 8-5
Current source gain-bandwidth

Range	SMU feedback impedance	60kHz ratio (DUT / SMU impedance)	Minimum DUT impedance
1nA	1GΩ	0.5	2GΩ
10nA	120MΩ	0.5	60MΩ
100nA	40MΩ	0.5	20MΩ
1µA	1.2MΩ	0.5	600kΩ
10µA	400kΩ	0.5	200kΩ
100µA	12kΩ	0.5	6kΩ
1mA	4kΩ	0.5	2kΩ
10mA	120Ω	0.5	60Ω
100mA	40Ω	0.5	20Ω
1A	1Ω	6	6Ω
1.5A	1Ω	6	6Ω
3A	0.3Ω	5	1.5Ω

Section 9

System Expansion (TSP-Link)

In this section:

Topic	Page
Overview	**9-2**
Master and Slaves	9-2
System configurations	9-2
Connections	**9-2**
Initialization	**9-3**
Assigning node numbers	9-3
Resetting the TSP-Link	9-4
Using the expanded system	**9-5**
Accessing nodes	9-5
System behavior	9-5
Triggering with TSP-Link	9-6
TSP advanced features	9-6
Using groups to manage nodes on the TSP-Link network	9-9
Running parallel test scripts	9-10
Using the data queue for real-time communication	9-11
Copying test scripts across the TSP-Link network	9-12
Removing stale values from the reading buffer	9-12

Overview

The Keithley Instruments Series 2600 System SourceMeter® 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 16 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 versus 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.

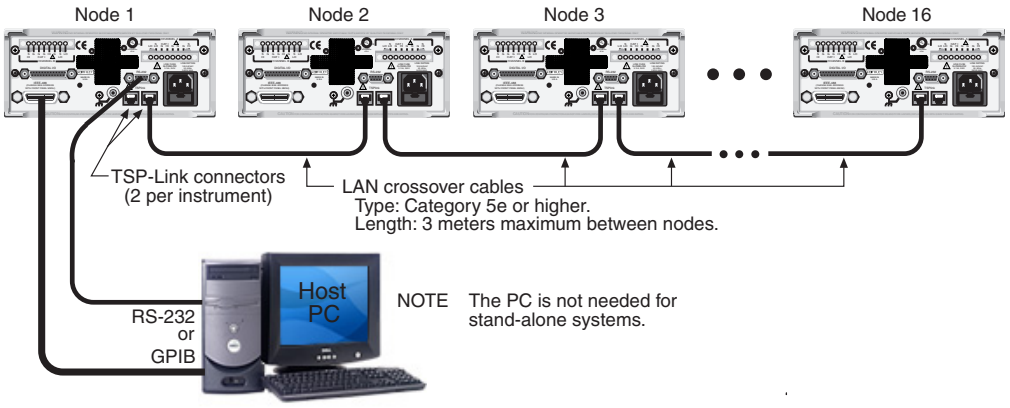
PC-based system – In a PC-based system, the GPIB or RS-232 interface to any single node becomes the interface to the entire system. In Figure 9-1, the system can be controlled via the GPIB or RS-232 interface of Node 1.

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 Series 2600 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

You can use the front panel of a instrument to assign a node number to that instrument (node). You can assign any number from 1 to 16 to the node.

Complete the following steps to assign a node number from the front panel of the instrument.

1. Press **Menu** > **TSPLINK** > **NODE**.
2. Use the navigation wheel to select the desired node number.
3. Press in on the navigation wheel and then turn the wheel left or right to select the desired number.
4. Press **ENTER** or push in on the navigation wheel to select the node number.

Remote programming

The `tsplink.node` attribute is used to set the node number for an instrument:

```
tsplink.node = N
```

where: `N = 1 to 16`

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.

NOTE 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:

Powering down or rebooting any unit in the system.

Rearranging or disconnecting the LAN cable connections between units.

Front panel operation

Complete the following steps to reset the TSP-Link network from the front panel.

1. Press **Menu** > **TSPLINK.**
2. Press **RESET.**

Remote programming

The commands associated with TSP-Link reset are listed in Table 9-1.

Table 9-1
TSP-Link reset commands

Command	Description
`tsplink.reset()`	Initializes the TSP-Link network.
`tsplink.state`	Returns `online` if the most recent TSP-Link reset was successful. Returns `offline` 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. Each entry is indexed by the node number of the unit. The variable for node [N] (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" in Section 12 for more details). SMU A on Node 1, therefore, could be accessed as `node[1].smua`.

The variable `localnode` is an alias for `node[N]`, where `N` is the node number of the Master. For example, if Node 1 is the Master, `localnode` can be used instead of `node[1]`.

NOTE You cannot access scripts stored on remote slave nodes.

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.

Triggering with TSP-Link

TSP-Link has three synchronization lines that function similar to the Digio synchronization lines. see “TSP-Link Synchronization lines” on page 10-10 for more information.

TSP advanced features

Use the TSP advanced features to run test scripts in parallel, to manage resources allocated to test scripts running in parallel, and to use the data queue to facilitate real-time communication between nodes on the TSP-Link network.

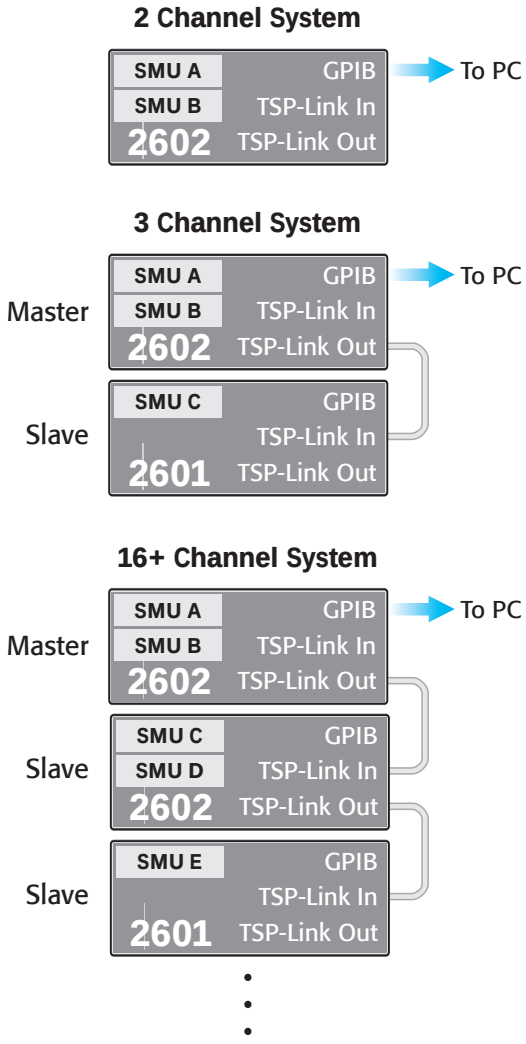
Running test scripts in parallel improves functional testing, provides higher throughput, and expands system flexibility.

There are two methods you can use to run test scripts in parallel:

- Create multiple TSP-Link networks
- Use a single TSP-Link network with groups

Figure 9-2 displays the first method, which consists of multiple TSP-Link networks. Each TSP-Link network has a master node and a GPIB connection to the PC.

Figure 9-2
Multiple TSP-Link networks



The second method to run parallel test scripts is to use groups with a single TSP-Link network. A group consists of one or more nodes with the same group number. Each group on the TSP-Link network can run different test scripts at the same time (in parallel).

Figure 9-3 displays a single TSP-Link network with groups. This method requires one TSP-Link network and a single GPIB connection to the PC.

Figure 9-3
Single TSP-Link network with groups

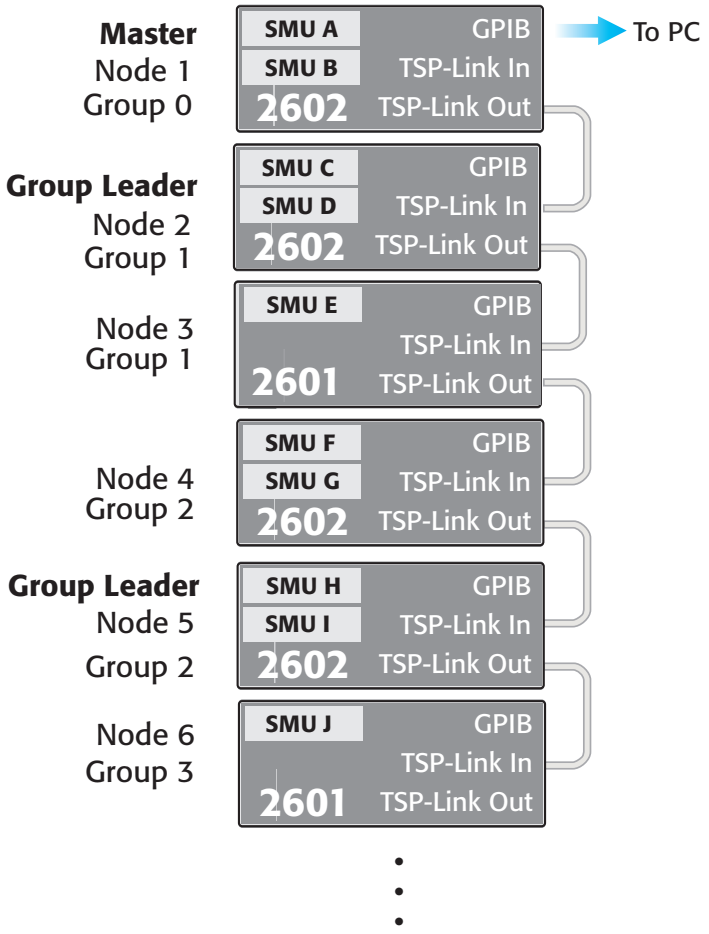


Table 9-2 describes the functions of a single TSP-Link network. Each group in this example runs multiple test scripts at the same time or in parallel.

Table 9-2:
TSP-Link network group functions

	Group members	Current function
0	• Master node	• Initiates and runs a test script on Node 2 • Initiates and runs a test script on Node 5 • Initiates and runs a test script on Node 6
	• Group leader Node 2	• Runs the test script initiated by the master node • Initiates remote operations on Node 3
	• Node 3	• Performs remote operations initiated by Node 2
	• Group leader Node 5	• Runs the test script initiated by the master node • Initiates remote operations on Node 4
	• Node 4	• Performs remote operations initiated by Node 5
	• Group leader Node 6	• Runs the test script initiated by the master node

Using groups to manage nodes on the TSP-Link network

The primary reason to use a group is to assign each node to run different test scripts at the same time (in parallel). Each node must belong to a group; a group can consist of one or more nodes on the TSP-Link network. Group numbers are not assigned automatically; you must use the Instrument Control Library (ICL) commands to assign each node to a group.

Master node overview

The master node is always the node that coordinates activity on the TSP-Link network. All nodes assigned to group 0 belong to the same group as the master node.

The following list describes the functionality of the master node:

- The only node that can issue the `execute` command to a remote node
- Cannot initiate remote operations on any node in a remote group if any node in that remote group is performing an overlapped operation
- Can use the `waitcomplete` command to wait for all overlapped operations running on the local group that the master node belongs to, and to wait for all overlapped operations running on a remote group, or to wait for all overlapped operations running on the TSP-Link network to complete

Group leader overview

Each group has a dynamic group leader. The last node in a group running any operation initiated by the master node is the group leader.

The following list describes the functionality of the group leader:

- Runs operations initiated by the master node
- Initiates remote operations on any node with the same group number
- Cannot initiate remote operations on any node with a different group number
- Can use the `waitcomplete` command without a parameter to wait for all overlapped operations running on nodes in the same group

Assigning groups

Group numbers can range from 0 (zero) to 64. The default group number is 0. You can change the group number at any time.

Use the following code to dynamically assign nodes to a group.

Note the following:

- Replace N with the node number
- N represents the node number that runs the test scripts and the TSL code
- Each time the node powers off, the group number for that node changes to 0
- Replace G with the group number

```
node[N].tsplink.group = G      -- Assigns the node to a group.
```

Reassigning groups

Use the following code to change group assignment. You can add or remove a node to a group at anytime.

```
node[N].tsplink.group = G      -- Assigns the node to a different group.
```

Running parallel test scripts

You can issue the `execute` command from the master node to initiate test script and TSL code on a remote node. The `execute` command places the remote node in the overlapped operation state. As a test script runs on the remote node, the master node continues to process other commands in parallel.

Note the following:

- Use the following code to send the `execute` command on a remote node
- N represents the node number that runs the test script
- Replace N with the node number

To set the global variable on Node N equal to 2.5:

```
node[N].execute("setpoint = 2.5")
```

The following code is an example of how to run a test script on a remote node.

NOTE For this example, `myscript` is defined on the local node.

To run myscript on Node N:

```
node[N].execute(myscript.source)
```

The following code demonstrates how to run a test script defined on a remote node.

NOTE For this example, `myscript` is defined on the remote node.

To execute a script defined on the remote node:

```
node[N].execute("myscript()")
```

It is recommended that you copy large scripts to a remote node to improve system performance. *see "Copying test scripts across the TSP-Link network" on page 9-12 for more information.*

Coordinating overlapped operations in remote groups

Errors occur if you send a command to a node in a remote group running an overlapped operation. All nodes in a group must be in the overlapped idle state before the master node can send a command to the group.

Use the `waitcomplete` command to:

- **Group leader and master node**: To wait for all overlapped operations running in the local group to complete
- **Master node only**: To wait for all overlapped operations running on a remote group to complete
- **Master node only**: To wait for all groups to complete overlapped operations

For additional information, reference the Instrument Control Library command `waitcomplete`.

The following code is an example on how to issue the `waitcomplete` command from the master node:

```
-- Waits for each node in group G to complete all overlapped operations.
waitcomplete(G)

-- Waits for all groups on the TSP-Link network to complete overlapped operations.
waitcomplete(0)
```

The group leader can issue the waitcomplete command to wait for the local group to complete all overlapped operations.

The following code is an example of how to issue the waitcomplete command:

```
-- Waits for all nodes in a local group to complete the overlapped operations.
waitcomplete()
```

Using the data queue for real-time communication

You cannot access the reading buffers or global variables from any node in a remote group while a node in that group is performing an overlapped operation. You can use the data queue to retrieve data from any node in a group performing an overlapped operation. In addition, the master node and the group leaders can use the data queue as a way to coordinate activities.

The data queue uses the first-in, first-out (FIFO) structure to store data. Nodes running test scripts in parallel can store data in the data queue for real-time communication. Each Series 2600 has an internal data queue. You can access the data queue from any node at any time.

You can use the data queue to post numeric values, strings, and tables. Tables in the data queue consume one entry. A new copy of the table is created when the table is retrieved from the data queue. The copy of the table does not contain any references to the original table or any subtables.

To add or retrieve values from the data queue and to view the capacity, *see "Instrument Control Library" on page 12-1 for more information.*

Copying test scripts across the TSP-Link network

To run a large script on a remote node, it is highly recommend that you copy the test script to the remote node to increase the speed of test script initiation.

Use the code below to copy test scripts across the TSP-Link network. This example creates a copy of a script on the remote node.

Note the following:

- The copy of the test script has the same name as the source
- Replace N with the number of the node that receives a copy of the script
- Replace `myscript` with the name of the script that you want to copy from the local node

```
-- Adds the source code from myscript to the data queue.
node[N].dataqueue.add(myscript.source)

-- Creates a new script on the remote node using the source code from myscript.
node[N].execute(myscript.name..“ = script.new(dataqueue.next(),
[[“..myscript.name..“]])“)
```

Removing stale values from the reading buffer

The node that acquires the data stores the data for the reading buffer. To optimize data access, all nodes can cache data from the node that stores the reading buffer data.

Running TSL code remotely can cause values in the reading buffer cache to become stale. If the values in the reading buffer change while the TSL code runs remotely, another node can hold stale values. Use the `clearcache` command to clear the cache.

The following code demonstrates how stale values occur and how to use the `clearcache` command to clear the reading buffer cache.

Note the following:

- Replace N with the node number
- Replace G with the group number

```
-- Creates a reading buffer on a node in a remote group.
node[N].tsplink.group = G
node[N].execute("rbremote = smua.makebuffer(20) " ..
                "smua.measure.count = 20  " ..
                "smua.measure.overlappedv(rbremote)")
waitcomplete(G)

-- Creates a variable on the local node to access the reading buffer.
rblocal = node[N].getglobal("rbremote")

-- Access data from the reading buffer.
print(rblocal[1])

-- Runs code on the remote node that updates the reading buffer.
node[N].execute("smua.measure.overlappedv(rbremote)")

-- Use the clearcache command if the reading buffer contains cached data.
rblocal.clearcache()
```

```
-- If you do not use the clearcache command, the data buffer values do not update. The same data
-- buffer values will print each time the print command is issued.
print(rblocal[1])
```

This page left blank intentionally.

Section 10

Digital I/O and Triggering

In this section:

Topic	Page
Overview	**10-2**
Digital I/O port	**10-2**
Port configuration	10-2
Digital I/O configuration	10-3
Controlling digital I/O lines	10-4
Output Enable (Models 2601/2602)	**10-7**
Overview	10-7
Operation	10-7
Front panel control of Output Enable	10-8
Remote control of Output Enable	10-8
Interlock (Models 2612/2612/2635/2636)	**10-8**
Overview	10-8
TSP-Link Synchronization lines	**10-10**
Connecting to TSP-Link	10-10
Digital I/O	10-10
Remote TSP-Link synchronization line commands	10-10
Triggering	**10-12**
Triggering types	10-12
Measurement triggering	10-12
Front panel triggering	10-13
Remote triggering	10-14
Hardware trigger modes	10-15
Understanding synchronous triggering modes	10-20

Overview

The documentation in this section provides detailed information on using the Digital I/O port and includes the following:

- "Digital I/O port"
- "Output Enable (Models 2601/2602)"
- "Interlock (Models 2612/2612/2635/2636)"

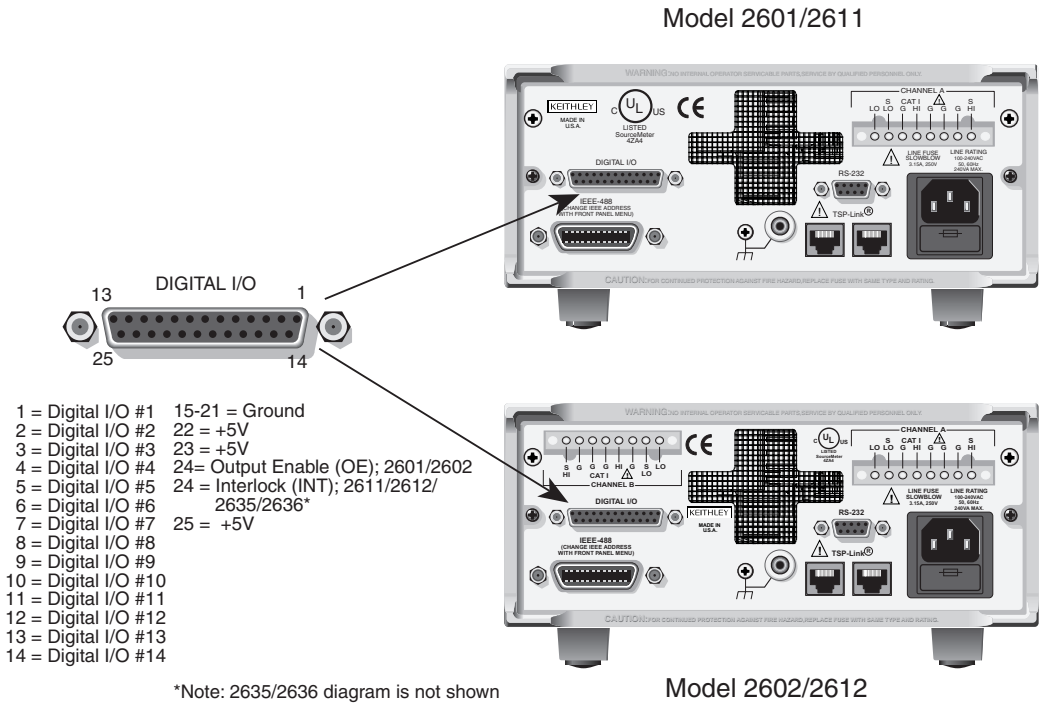
Digital I/O port

The Keithley Instruments Series 2600 System 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.

Figure 10-1
Digital I/O port



Connecting cables

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

Digital I/O lines

The port provides 14 digital I/O lines. Each output is set high (+5V) or low (0V) and can read high or low logic levels. Each digital I/O line is an open-drain signal.

+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).

Output Enable and Interlock line

The Model 2601/2602 OE (output enable) line and the Model 2611/2612/2635/2636 INT (Interlock) 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 (Models 2601/2602)" for Model 2601/2602 operation details, or "Interlock (Models 2612/2612/2635/2636)" for Model 2611/2612/2635/2636 operation details, described later in this section.

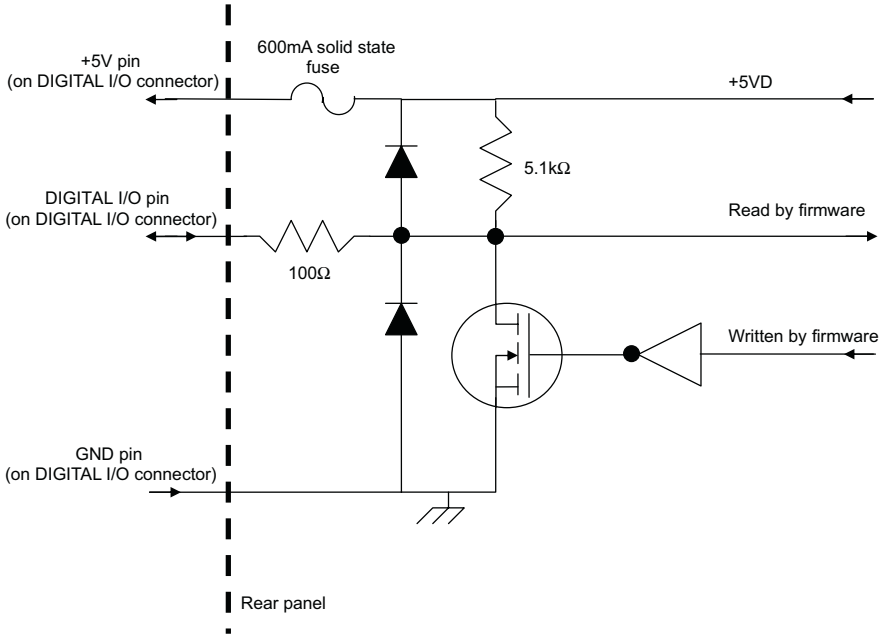
WARNING ***The Digital I/O port of the Model 2601/2602 SourceMeter is not suitable for control of safety circuits and should not be used to control a safety interlock. The Model 2611/2612/2635/2636 SourceMeter Digital I/O port includes an Interlock line to be used as safety interlock. When an interlock is required for safety, a separate circuit should be provided that meets the requirements of the application to reliably protect the operator from exposed voltages.***

Digital I/O configuration

Figure 10-2 shows the basic configuration of the Digital I/O port. Writing a 1 to a line sets that line high (~ +5V). Writing a 0 to a line sets that line low (~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 480µA per I/O line.

Figure 10-2
Digital I/O port configuration

DIGITAL I/O INTERFACE (All Series 2600 Models):
Connector: 25-pin female D
Input/Output pins: 14 open-drain I/O bits
Absolute maximum input voltage: 5.25V
Absolute minimum input voltage: -0.25V
Maximum logic low input voltage: 0.7V @ +850µA
Minimum logic high input voltage: 2.1V @ +570µA
Maximum source current (flowing out of digital I/O bit): +960µA
Absolute Maximum sink current (flowing into digital I/O bit): -11.0A
Maximum Sink Current @ Maximum Logic Low Voltage (0.7V): -5.0mA.



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 interface, 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

Setting digital I/O values

To set digital I/O values:

1. Press the MENU > DIGOUT, and then press ENTER or push in the navigation wheel.
2. Select DIG-IO-OUTPUT, and then press ENTER of the navigation wheel.
3. 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 navigation wheel.
4. 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:

1. Press MENU > DIGOUT and then press ENTER or the navigation wheel.
2. Select WRITE-PROTECT, then press ENTER or the navigation wheel.
3. 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 navigation wheel.
4. Press EXIT as needed to return to the normal display.
5. 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.

Table 10-2
Digital I/O commands

Command	Description
Commands for basic I/O:	
digio.readbit(bit)	Read one digital I/O input line
digio.readport()	Read digital I/O port
digio.writebit(bit, data)	Write data to one digital I/O output line
digio.writeport(data)	Write data to digital I/O port
digio.writeprotect = mask	Write protect mask to digital I/O port
Commands for digital I/O triggering:	
digio.trigger[line].assert()	Generate a trigger on digital I/O line
digio.trigger[line].clear()	Clear the event detector for a trigger
digio.trigger[line].mode = mode	Control I/O trigger event detector mode: see “digio.trigger[N].mode” on page 12-20 for more information.
digio.trigger[line].pulsewidth = width	Set trigger pulse width
digio.trigger[line].release()	Release latched trigger
digio.trigger[line].wait(timeout)	Wait for a trigger

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 Series 2600 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.

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 10µs, 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 10ms.
digio.trigger[2].mode = digio.TRIG_FALLING  --Set line 2 mode to falling edge.
digio.trigger[2].assert()                   --Assert trigger on line 2.
```

Output Enable (Models 2601/2602)

Overview

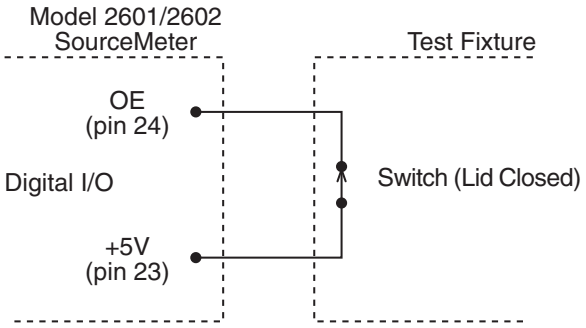
The Model 2601/2602 Digital I/O Port provides an Output Enable line for use with a test fixture switch. When properly used, the output of the SourceMeter will turn OFF when the lid of the test fixture is opened. See Section 3 for important safety information when using a test fixture.

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 Model 2601/2602 SourceMeter is not suitable for control of safety circuits and should not be used to control a safety interlock.***

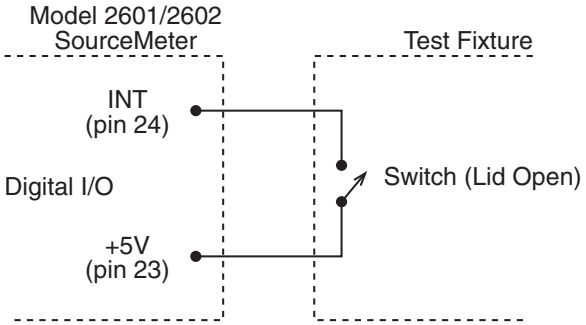
Operation

When enabled, the output of the Model 2601/2602 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 +5V is applied to the Output Enable line.

Figure 10-3
Using Model 2601/2602 Output Enable



A. OUTPUT can be turned on.



B. OUTPUT cannot be turned on.

Front panel control of Output Enable

To activate the Output Enable line:

1. Press the CONFIG key followed by the OUTPUT key.
2. Choose DIO-CONTROL, then press ENTER or the navigation wheel.
3. Select OE_OUTPUT_OFF to activate the Output Enable signal causing the SMU output to be blocked if the Output Enable is not asserted (connect to +5V). Select NONE to deactivate the Output Enable signal so that its state has no effect on the SMU output.
4. Press EXIT as needed to return to the normal display.

Remote control of Output Enable

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 Model 2601/2602 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 activates the Output Enable for Channel A:

```
smua.source.outputenableaction = smua.OE_OUTPUT_OFF
```

Interlock (Models 2612/2612/2635/2636)

Overview

The Model 2611/2612/2635/2636 Digital I/O Port provides an Interlock 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.

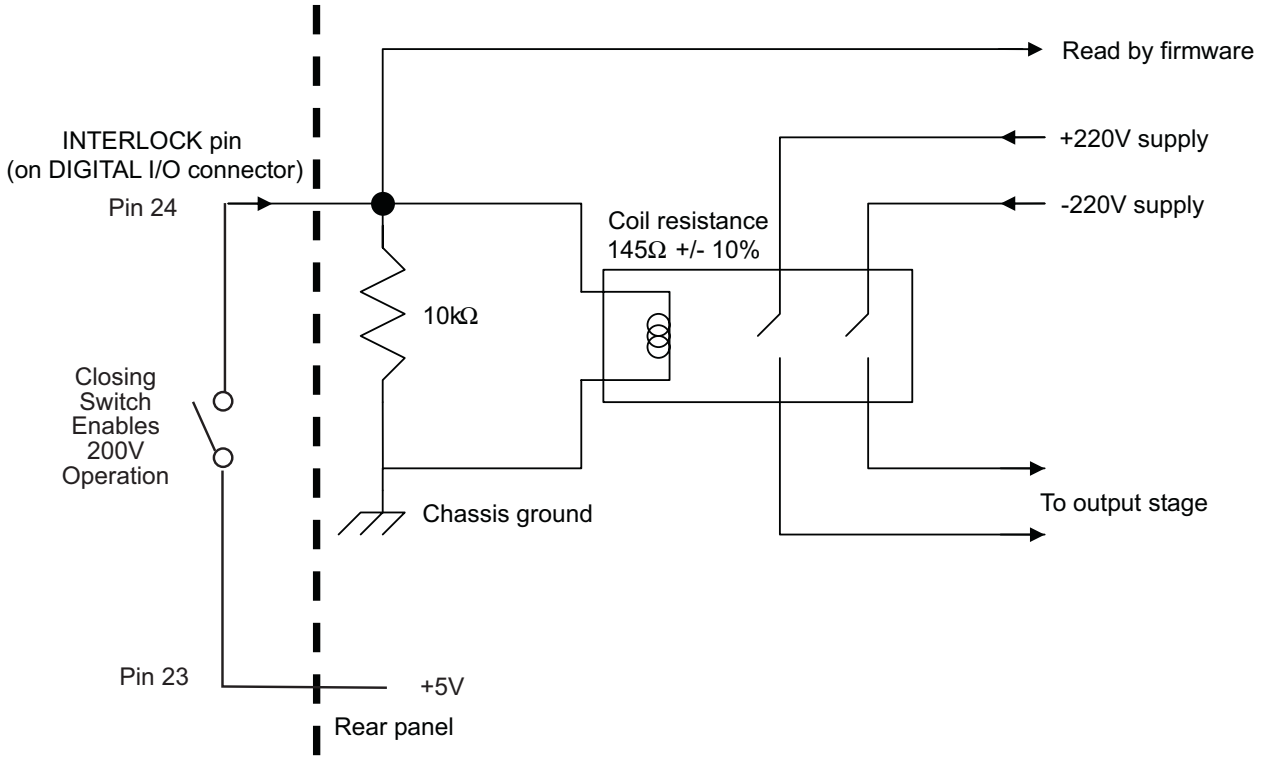
CAUTION	If the Interlock line is switched excessively (more than 10,000 times), its reliability may be reduced. Where the Interlock is used for safety, it should be serviced regularly to ensure proper operation.

Operation

When on the 200V source range, the output of the Model 2611/2612/2635/2636 SourceMeter can only be turned on when the Interlock line is pulled high through a switch to +5V, as shown in Figure 10-4. If the lid of the test fixture opens, the switch opens, and the Interlock line goes low, turning the output of the Model 2611/2612/2635/2636 SourceMeter off. The output will not be automatically turned on when the Interlock line is set high. The output cannot be turned back on until the Interlock line is set high.

A signal of > 3.4V at 24mA (absolute maximum of 6V) must be externally applied to this pin to ensure 200V operation. This signal is pulled down to chassis ground with a 10kΩ resistor. 200V operation will be blocked when the INTERLOCK signal is < 0.4V (absolute minimum of -0.4V).

Figure 10-4
Using Model 2611/2612/2635/2636 **Interlock**



TSP-Link Synchronization lines

The Series 2600 System SourceMeter® has three synchronization lines that you can use for triggering, digital I/O, and to synchronize multiple instruments on a TSP-Link network.

Connecting to TSP-Link

The TSP-Link synchronization lines are built into TSPLink. Use the TSP-Link connectors located on the back of the Series 2600 SourceMeter. If you use the TSP-Link network, you do not have to modify your connections. See, "System Expansion (TSP-Link)," page 9-1 for detailed information about connecting to TSP-Link.

Digital I/O

Each synchronization line is an open-drain signal. When using the TSP-Link synchronization lines for digital I/O, any node that sets the programmed line state to 0 causes all nodes to read 0 from the line state. This occurs regardless of the programmed line state of any other node.

Digital I/O bit weighting

Table 10-3 displays the bit weighting for the digital I/O lines.

Table 10-3
Digital I/O bit weighting

Line #	Bit	Decimal weighting	Hexadecimal weighting
1	B1	1	0x0001
2	B2	2	0x0002
3	B3	4	0x0004

Remote TSP-Link synchronization line commands

Commands that control and access the TSP-Link synchronization port are summarized in Table 10-4. See Section 12 for complete details on these commands. See Table 10-3 for the decimal and hexadecimal values used to control and access the digital I/O port and individual lines.

Table 10-4
TSP-Link Triggering commands

Command	Description
Commands for basic I/O:	
tsplink.readbit(bit)	Read one digital I/O input line
tsplink.readport()	Read digital I/O port
tsplink.writebit(bit, data)	Write data to one digital I/O output line
tsplink.writeport(data)	Write data to digital I/O port
tsplink.writeprotect = mask	Write protect mask to digital I/O port

Table 10-4
TSP-Link Triggering commands

Command	Description
Commands for digital I/O triggering:	
tsplink.trigger[N].assert()	Generates a trigger on the synchronization line
tsplink.trigger[N].clear()	Clear the event detector for a trigger
tsplink.trigger[N].mode = mode	Control I/O trigger event detector mode: see "tsplink.trigger[N].mode" on page 12-113 for more information.
tsplink.trigger[line].pulsewidth = width	Sets the trigger pulse width
tsplink.trigger[line].release()	Release the latched trigger
tsplink.trigger[line].wait(timeout)	Waits for a trigger

Basic digital I/O commands

Use the following commands to perform basic steady-state digital I/O operations such as reading and writing to individual I/O lines or reading and writing to the entire port.

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

TSP-Link Synchronization line trigger commands

Use the following commands to trigger the Series 2600 using TSP-Link synchronization lines.

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.

```
tsplink.writebit(1,1)                    --Set bit 1 high.
data = tsplink.readport()                --Read digital I/O port.
```

Synchronization line trigger example

The commands below set the line 2 pulse width to 10µs, trigger mode to falling edge, and then assert a trigger pulse on that synchronization line:

```
tsplink.trigger[2].pulsewidth = 1e-5                - Set line 2 pulse width to 10ms.
tsplink.trigger[2].mode = tsplink.TRIG_FALLING      - Set line 2 mode to falling edge.
tsplink.trigger[2].assert()                         - Assert trigger on line 2.
```

Triggering

Triggering types

A trigger initiates an event within the Series 2600 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 Series 2600 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).

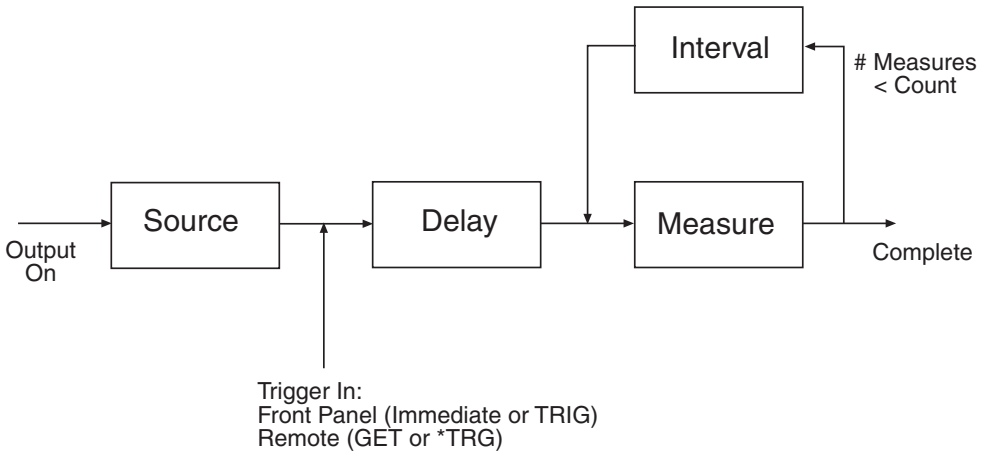
NOTE It is not necessary to change any trigger settings to use the basic source and measurement procedures covered in this section. Simply make sure that the unit is reset to the factory default conditions by using the MENU > SETUP > RECALL > FACTORY option before using those procedures.

Measurement triggering

Figure 10-5 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 10-5
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 navigation wheel.
3. Select MANUAL, then press ENTER or the navigation wheel.
4. Choose COUNT, then select INFINITE, and press ENTER or the navigation wheel.
5. Select INTERVAL, set the interval to 1s, then press ENTER or the navigation wheel.

6. Choose DELAY, set the delay to 2s, then press ENTER.
7. Press EXIT to return to normal display.
8. 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.
9. Turn the OUTPUT off to stop taking readings.

Remote triggering

Remote trigger commands

Trigger commands are listed in Table 10-3. 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.

It is important to note the following:

- A synchronization line event detector sets the trigger overrun if an edge is detected while the event detector is in the detected state
- You can use the `digo.trigger[N].clear` command to retest the event detector detected state and to clear the trigger overrun indication
- Review the trigger overrun to confirm that a trigger was not missed

Command	Description
trigger.clear() triggered = trigger.wait(timeout)	Clears triggers pending. Waits the timeout 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. A maximum of 50 *TRG commands can be used before the input buffer overflows.

```
smua.reset()                              - Restore 2600 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.
```

Hardware trigger modes

Use the hardware trigger modes to integrate Keithley Instruments and non-Keithley instruments into an efficient test system. The hardware synchronization lines are classic trigger lines. The Series 2600 contains 14 digital I/O lines and three TSP-Link synchronization lines that you can use for input or output triggering. Table 10-5 provides a summary for each hardware trigger mode.

Table 10-5
Hardware trigger mode summary

	Output		Input	Notes
	Unasserted	Asserted	Detects	
	N/A	N/A	N/A	Use the `writebit` and `writeport` commands for direct line control (Version 1.4.0 and higher).
Either Edge	High	Low	Either	Short input pulses can cause a trigger overrun.
Falling Edge	High	Low	Falling	
Rising Edge	N/A	N/A	N/A	• The programmed state of the line determines if the behavior is similar to RisingA or RisingM •High similar to RisingA •Low similar to RisingM
	High	Low	Rising	
RisingM	Low	High	None	
	High latching	Low	Falling	• Behaves similar to SynchronousA • Trigger overrun detection is disabled • To mirror the SynchronousA trigger mode, set the pulse duration to 1µs or any small nonzero value
SynchronousA	High latching	High	Falling	Ignores the pulse duration.
SynchronousM	High	Low	Rising	

Each trigger mode controls the input trigger detection and output trigger generation. The input detector monitors for and detects all edges, even if the node that generates the output trigger causes the edge.

A trigger overrun generates if an input trigger is received before the previous input trigger processes. To determine if a trigger overrun has occurred, reference the trigger overrun attributes.

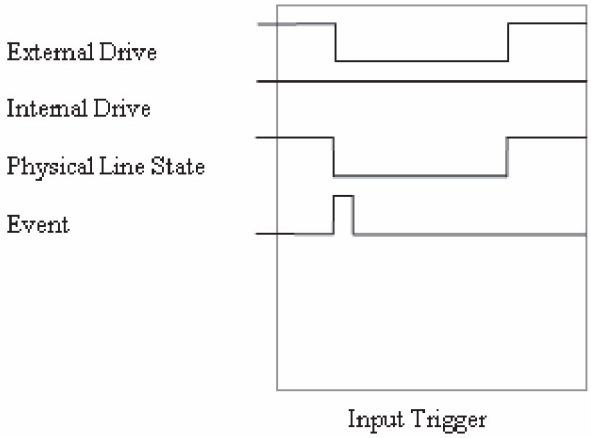
For additional information on the hardware trigger modes, *see "Instrument Control Library" on page 12-1 for more information.*

NOTE To have direct control of the line state, use the Bypass trigger mode.

Falling Edge trigger mode

The Falling Edge trigger mode generates low pulses and detects all falling edges. Figure 10-6 illustrates the characteristics of the falling edge input trigger.

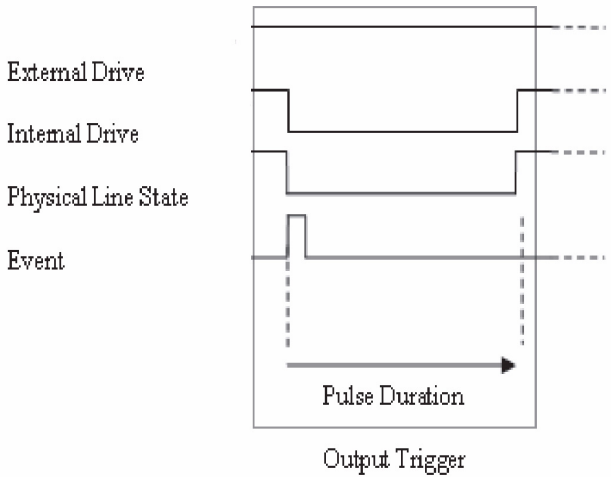
Figure 10-6: Falling Edge input trigger



Input characteristics:

- Detects all falling edges as input triggers

Figure 10-7: Falling Edge output trigger



Output characteristics

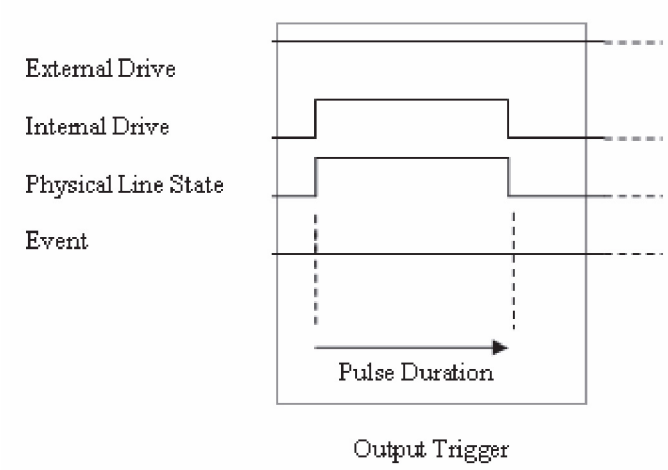
- The `trigger.assert` command generates a low pulse for the programmed pulse duration

Rising Edge master trigger mode (version 1.4.0 or higher)

Use the Rising Edge master trigger mode (RisingM) to synchronize with non-Keithley instruments that require a high pulse. Input trigger detection is not available in this trigger mode. You can use the RisingM trigger mode to generate rising edge pulses.

NOTE The RisingM trigger mode does not function properly if the line is driven low by an external drive.

Figure 10-8: RisingM output trigger



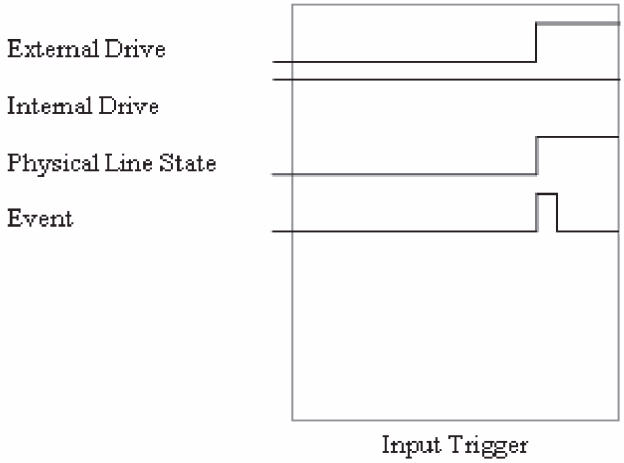
Output characteristics:

- The `trigger.assert` command causes the physical line state to float high during the trigger pulse duration

Rising Edge Acceptor trigger mode (version 1.4.0 or higher)

The Rising Edge Acceptor trigger mode (RisingA) generates a low pulse and detects rising edge pulses. Figure 10-9 displays the RisingA input trigger.

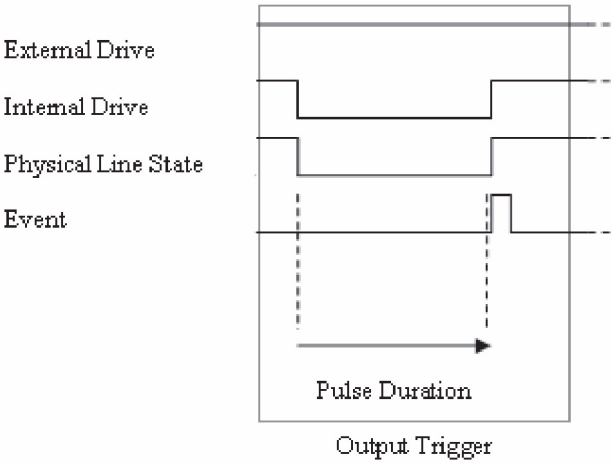
Figure 10-9: RisingA input trigger



Input characteristics:

- All rising edges generate an input event

Figure 10-10 RisingA output trigger



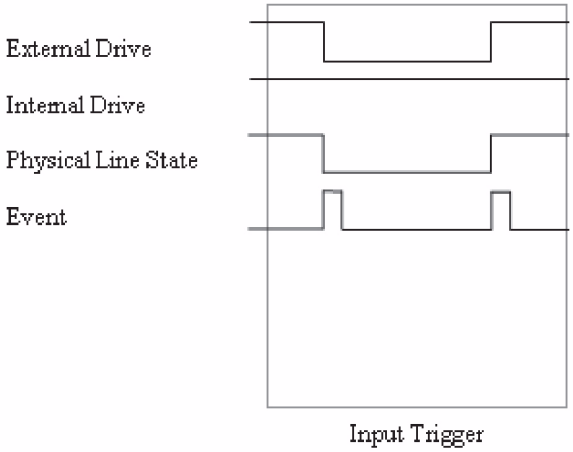
Output characteristics:

- The `trigger.assert` command generates a low pulse that is similar to the Falling Edge trigger mode

Either Edge trigger mode

The Either Edge trigger mode generates a low pulse and detects both rising and falling edges.

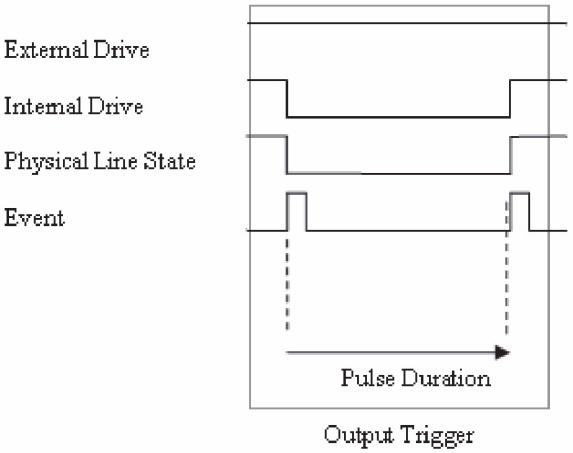
Figure 10-11: Either Edge input trigger



Input characteristics:

- All rising or falling edges generate an input trigger event

Figure 10-12: Either Edge output trigger



Output characteristics:

- The `trigger.assert` command generates a low pulse that is similar to the Falling Edge trigger mode

Understanding synchronous triggering modes

Use the synchronous triggering modes to implement bidirectional triggering, to wait for one node, or to wait for a collection of nodes to complete all triggered actions.

All non-Keithley instrumentation must have a trigger mode that functions similar to the SynchronousA or SynchronousM trigger modes.

To use synchronous triggering, configure the triggering master to SynchronousM trigger mode or the non-Keithley equivalent. Configure all other nodes in the test system to SynchronousA trigger mode or a non-Keithley equivalent.

Synchronous master trigger mode

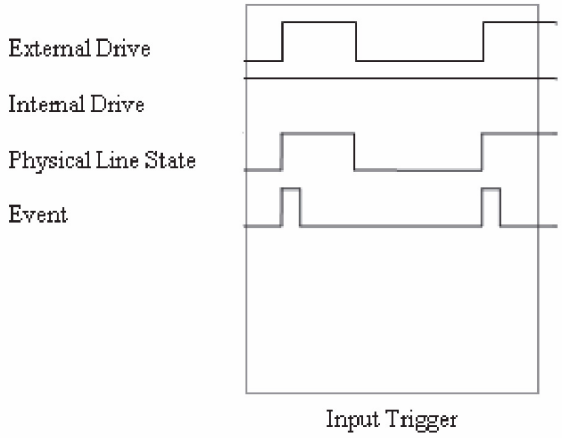
Use the Synchronous master trigger mode (SynchronousM) to generate falling edge output triggers, to detect the rising edge input triggers, and to initiate an action on one or more external nodes with the same trigger line.

In this mode, the output trigger consists of a low pulse. All non-Keithley instruments attached to the synchronization line in a trigger mode equivalent to SynchronousA must latch the line low during the pulse duration.

To use the SynchronousM trigger mode, configure the triggering master as SynchronousM and then configure all other nodes in the test system as Synchronous, SynchronousA, or to the non-Keithley equivalent.

NOTE Use the SynchronousM trigger mode to receive notification when the triggered action on all nodes is complete.

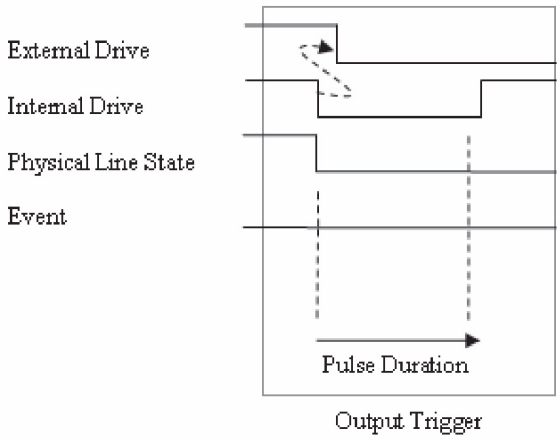
Figure 10-13: SynchronousM input trigger



Input characteristics:

- All rising edges are input triggers
- When all external drives release the physical line, the rising edge is detected as an input trigger
- A rising edge cannot be detected until all external drives release the line and the line floats high

Figure 10-14: SynchronousM output trigger



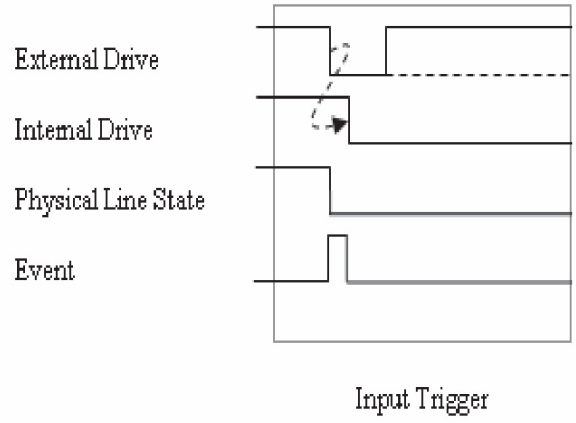
Output characteristics:

- The `trigger.assert` command generates a low pulse that is similar to the Falling Edge trigger mode

Synchronous Acceptor trigger mode

Use the Synchronous Acceptor trigger mode (SynchronousA) in conjunction with the SynchronousM trigger mode. The role of the internal and external drives are reversed in the SynchronousA trigger mode.

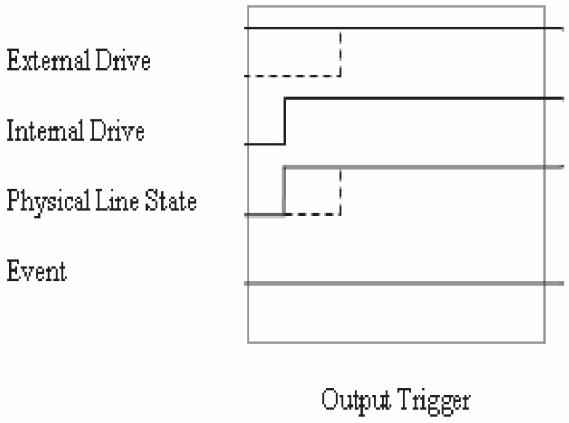
Figure 10-15: SynchronousA input trigger



Input characteristics:

- The falling edge is detected as the external drive pulses the line low, and the internal drive latches the line low

Figure 10-16: SynchronousA output trigger



Output characteristics:

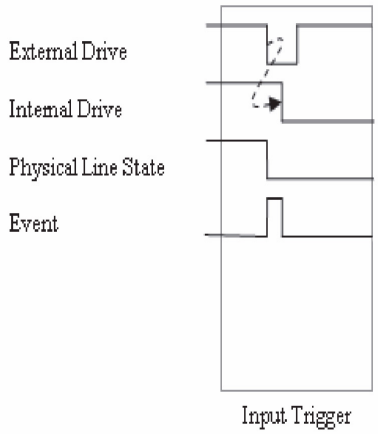
- The `trigger.assert` command releases the line if the line is latched low
- The physical line state does not change until all drives (internal and external) release the line

Synchronous trigger mode

The Synchronous trigger mode is a combination of SynchronousA and SynchronousM trigger modes. Use the Synchronous trigger mode for backwards firmware compatibility.

The SynchronousA and SynchronousM trigger modes provide additional flexibility. It is recommended that you use SynchronousA and SynchronousM for firmware v1.4.0 or higher, and use the Synchronous trigger mode for firmware prior to v1.4.0.

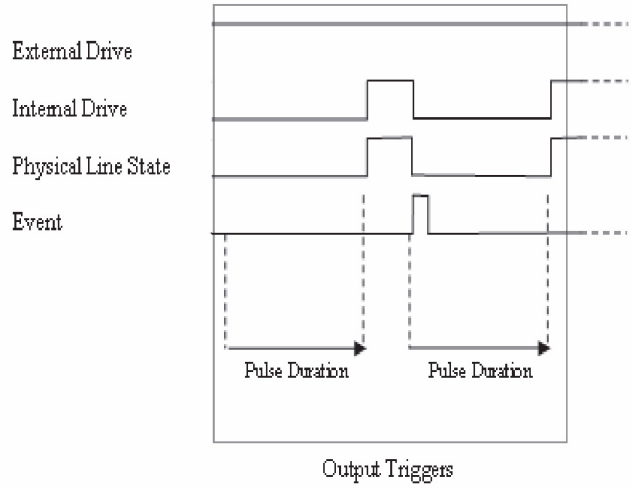
Figure 10-17: Synchronous input trigger



Input characteristics:

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

Figure 10-18: Synchronous output trigger



Output characteristics:

- The `trigger.assert` command generates a low pulse for the programmed pulse duration if the line is latched low, a falling edge does not occur
- A normal falling edge pulse generates when the internal drive is not latched low and the `trigger.assert` command is issued

This page left blank intentionally.

Section 11

Communications Interfaces

In this section:

Topic	Page
Overview	**11-2**
Selecting an interface	**11-2**
GPIB operation	**11-2**
GPIB standards	11-2
GPIB connections	11-2
Primary address	11-4
Terminator	11-5
General bus commands	**11-6**
REN (remote enable)	11-6
IFC (interface clear)	11-6
LLO (local lockout)	11-6
GTL (go to local)	11-6
DCL (device clear)	11-6
SDC (selective device clear)	11-7
GET (group execute trigger)	11-7
SPE, SPD (serial polling)	11-7
Front panel GPIB operation	**11-7**
Error and status messages	11-7
GPIB status indicators	11-7
LOCAL key	11-8
RS-232 interface operation	**11-8**
Setting RS-232 interface parameters	11-8
Sending and receiving data	11-9
Terminator	11-9
Baud rate	11-9
Data bits and parity	11-10
Flow control (signal handshaking)	11-10
RS-232 connections	11-11
Error messages	11-11

Overview

The documentation in this section provides detailed information on using communications interfaces:

- "Selecting an interface"
- "GPIB operation"
- "General bus commands"
- "Front panel GPIB operation"
- "RS-232 interface operation"

Selecting an interface

The Keithley Instruments Series 2600 System SourceMeter® supports two built-in remote interfaces:

- GPIB (General Purpose Interface Bus)
- RS-232 interface

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.

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.

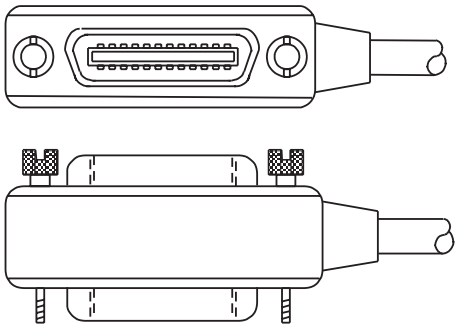
GPIB standards

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

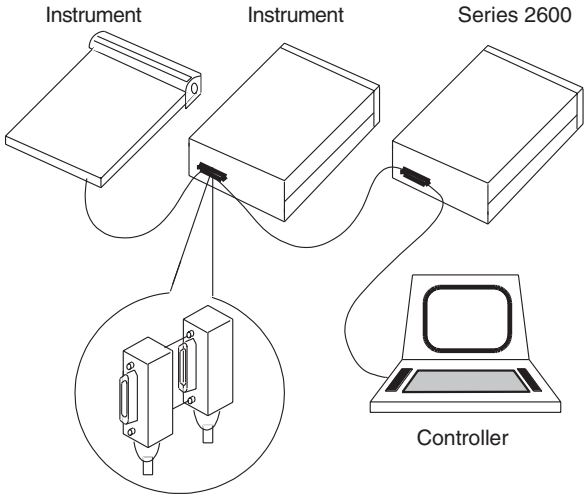
To connect the SourceMeter to the GPIB bus, use a cable equipped with standard IEEE-488 connectors, as shown in Figure 11-1.

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 connection scheme for a multi-unit test system.

Figure 11-2
IEEE-488 connections

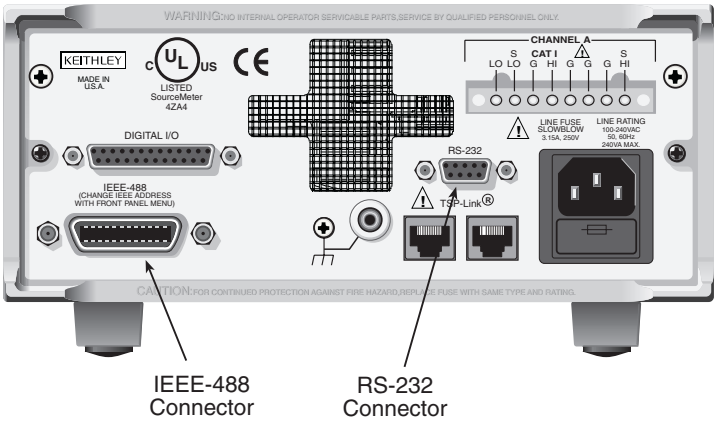


To avoid possible mechanical damage, stack no more than three connectors on any one unit. To minimize interference caused by electromagnetic radiation, use only shielded IEEE-488 cables. Available shielded cables from Keithley Instruments are listed in "Options and accessories" in Section 1.

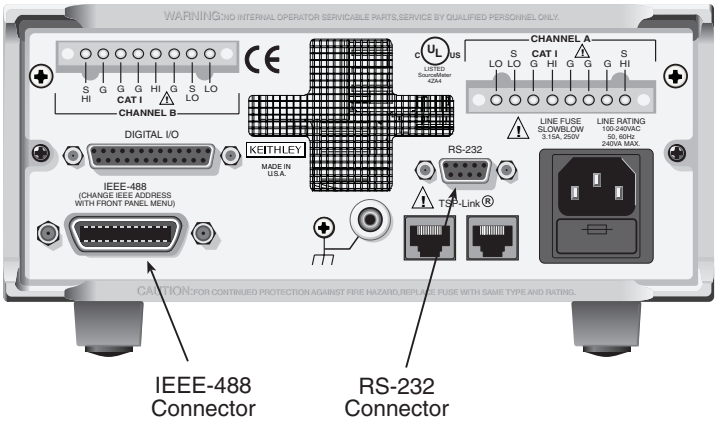
To connect the SourceMeter to the IEEE-488 bus, line up the cable connector with the connector located on the rear panel. Install and tighten the screws securely, making sure not to overtighten them (Figure 11-3 shows the location of the connections).

Figure 11-3
IEEE-488 and RS-232 connector locations

Model 2601/2611



Model 2602/2612



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

Primary address

The 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 MENU > GPIB, and then press ENTER or the rotary knob.
2. Set the primary address to the desired value, then press ENTER or the rotary knob.
3. 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 Series 2600 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 from a remote interface.

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 MENU > RS-232 and then press ENTER or push in on the the Rotary Knob.
2. Complete the followng interface parameters:
   - Baud rate
   - Number of bits
   - Parity
   - Flow control
     (See the following section for details)
3. 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.

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)

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.

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 using the serial.read() command, the Series 2600 will terminate on any line feed character. When sending data using the serial.write() command, it will *not* append a terminator. Be sure to append the appropriate terminator to the message before sending it.

Baud rate

The baud rate is the rate at which the 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

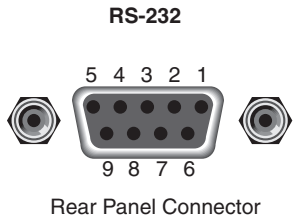
The RS-232 interface can be configured to send/receive data that is 7 or 8 bits long using even, odd, or no parity. No parity is only valid when using 8 data bits.

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 Series 2600 is ready to send (RTS) data, it will transmit when it receives the clear to send (CTS) signal from the computer.

Figure 11-4
RS-232 interface connector



Rear Panel 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

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.

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 Instruments is listed in ""Options and accessories"" in Section 1.

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 Series 2600 does not use all RS-232 signals. See Table 11-3.

Error messages

See Appendix B for RS-232 error messages.

This page left blank intentionally.

Section 12

Instrument Control Library

In this section:

Topic	Page
Command programming notes	**12-2**
Conventions	12-2
Functions and attributes	12-3
TSP-Link nodes	12-4
Logical instruments	12-5
Reading buffers	12-5
Time and date values,	12-7
ICL functions and attributes	**12-8**

Command programming notes

Conventions

For the following command reference, it is necessary to understand the following conventions:

Wild characters

Many Source-measure-unit (SMU) commands are expressed in a generic form using wild characters. A wild character indicates a 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
```

NOTE 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 Series 2600 to assert an output trigger on trigger line 5, the following command statement is sent to the instrument.

```
digio.trigger[5].assert()
```

NOTE The wild character N should NOT to 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” in following subsection).

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 a SMU. A function based command is not necessarily directly related to a Series 2600 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:

`digio.writeport(15)`	Sets digital I/O lines 1, 2, 3 and 4 high.
`digio.writebit(3, 0)`	Sets line 3 low (0).
`smua.reset()`	Returns SMU A to its default settings.
`digio.readport()`	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 a 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 predefined 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 Model 2601/2602 voltage source can be set from -40.4V to +40.4V, while the Model 2611/2612 voltage source can be set from -202V to +202V. 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 Test script program (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:

id	The node number assigned to the node.
model	The product model number string of the node.
revision	The product revision string of the node.
serialno	The product serial number string of the node.

There is also an entry for each logical instrument on the node (see “Logical instruments”).

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. For TSP and Instrument command library (ICL)., 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 Series 2600 SMU has the following logical instruments per enclosure:

beeper	errorqueue	smua	status
digio	gpib	smub	tsplink
display	serial	timer	

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:

name A string that represents the logical instrument's name. For example, `smua`.

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 9th 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`:

`appendmode`	Off 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 `rb[n+1]`. This attribute is initialized to off when the buffer is created.
`basetimestamp`	The time stamp of when the reading at `rb[1]` was stored, in seconds, from time of power-up.
`capacity`	The total number of readings that can be stored in the reading buffer.
`collectsourcevalues`	When on, source values will be stored with readings in the buffer. This requires four extra bytes of storage per reading. This value, off or on, can only be changed when the buffer is empty. When the buffer is created, this attribute is initialized to off.
`collecttimestamps`	When on, time stamps will be stored with readings in the buffer. This requires four extra bytes of storage per reading. This value, off or on, can only be changed when the buffer is empty. When the buffer is created, this attribute is initialized to off.
n	The number of readings in the reading buffer.
`timestampresolution`	The time stamp resolution, in seconds. When the buffer is created, its initial resolution is 0.000001 seconds. At this resolution, the reading buffer can store unique time stamps 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.

`measurefunctions`	An array (TSL table) of strings indicating the function measured for the reading (Current, Voltage, Ohms or Watts).
`measureranges`	An array (TSL table) of full scale range values for the measure range used when the measurement was made.
`readings`	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., `rb[2]` and `rb.readings[2]` are the same value.
`sourcefunctions`	An array (TSL table) of strings indicating the source function at the time of the measurement (Current or Voltage).
`sourceoutputstates`	An array (TSL table) of strings indicating the state of the source (Off or On).
`sourceranges`	An array (TSL table) of full scale range values for the source range used when the measurement was made.
`sourcevalues`	If enabled (see `collectsourcevalues` above), an array (TSL table) of the sourced value in effect at the time of the reading.
`statuses`	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.
`timestamps`	An array (TSL table) of time stamps, in seconds, of when each reading occurred. These can be compared to the base time stamp 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

Time and date 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 Series 2600 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

beeper function and attribute, page 12-10

beeper.beep
beeper.enable

bit functions, page 12-10

bit.bitor
bit.bitand
bit.bitxor
bit.clear
bit.get
bit.getfield
bit.set
bit.setfield
bit.toggle
bit.test

data queue functions 12-15

dataqueue.add
dataqueue.CAPACITY
dataqueue.clear
dataqueue.count
dataqueue.next

delay function, page 12-17

delay

digio functions and attributes, page 12-17

digio.readbit
digio.readport
digio.trigger[N].assert
digio.trigger[N].clear
digio.trigger[N].mode
digio.trigger[N].overrun
digio.trigger[N].pulsewidth
digio.trigger[N].release
digio.trigger[N].wait
digio.writebit
digio.writeport
digio.writeprotect

display functions and attributes, page 12-23

display.clear
display.getannunciators
display.getcursor
display.getlastkey
display.gettext
display.inputvalue
display.loadmenu.add
display.loadmenu.delete
display.locallockout
display.menu
display.numpad
display.prompt
display.screen
display.sendkey
display.setcursor
display.settext
display.smuX.digits
display.smuX.measure.func
display.trigger.clear
display.trigger.wait
display.waitkey

errorqueue functions and attribute, page 12-35

errorqueue.clear
errorqueue.count
errorqueue.next

exit function, page 12-36

exit

format attributes, page 12-36

format.asciiprecision
format.byteorder
format.data

gpib attribute, pg. 12-39

gpib.address

localnode attribute, page 12-39

localnode.linefreq
localnode.autolinefreq
localnode.showerrors

makegetter functions, page 12-42

makegetter
makesetter

opc function, page 12-43

opc

printbuffer and printnumber functions, page 12-43

printbuffer
printnumber

reset function, page 12-45

reset

serial functions and attributes, page 12-45

serial.baud
serial.databits
serial.flowcontrol
serial.parity
serial.read
serial.write

setup functions and attribute, page 12-48

setup.poweron
setup.recall
setup.save

smuX functions and attributes, page 12-48

smuX.cal.adjustdate
smuX.cal.date
smuX.cal.due
smuX.cal.lock
smuX.cal.password
smuX.cal.polarity
smuX.cal.restore
smuX.cal.save
smuX.cal.state
smuX.cal.unlock
smuX.contact.calibratelo
smuX.contact.calibratehi
smuX.contact.check
smuX.contact.r
smuX.contact.speed
smuX.contact.threshold
smuX.makebuffer

smuX.measure.analogfilter
smuX.measure.autorangeY
smuX.measure.autozero
smuX.measure.calibrateY
smuX.measure.count
smuX.measure.delay
smuX.measure.delayfactor
smuX.measure.filter.count
smuX.measure.filter.enable
smuX.measure.filter.type
smuX.measure.interval
smuX.measure.lowrangeY
smuX.measure.nplc
smuX.measure.overlappedY
smuX.measure.overlappediv
smuX.measure.rangeY
smuX.measure.rel.enableY
smuX.measure.rel.levelY
smuX.measure.Y
smuX.measure.iv
smuX.measureYandstep
smuX.measureivandstep
smuX.nvbufferY
smuX.nvbufferY.appendmode
smuX.nvbufferY.basetimestamp
smuX.nvbufferY.capacity
smuX.nvbufferY.clear
smuX.nvbufferY.collecttimestamps X = SMU channel (a or b)
smuX.nvbufferY.n
smuX.nvbufferY.timestampresolution X = SMU channel (a or b)
smuX.reset
smuX.sense
smuX.source.autorangeY
smuX.source.calibrateY
smuX.source.compliance
smuX.source.delay
smuX.source.func
smuX.source.levelY
smuX.source.limitY
smuX.source.lowrangeY
smuX.source.offlimiti
smuX.source.offmode
smuX.source.output
smuX.source.outputenableaction
smuX.source.rangeY

status function and attributes, page 12-76

status.condition
status.measurement.*
status.measurement.buffer_available.*
status.measurement.current_limit.*
status.measurement.instrument.*
status.measurement.instrument.smuX.*
status.measurement.reading_overflow.*
status.measurement.voltage_limit.*
status.node_event
status.node_enable
status.operation.user.*
status.operation.*
status.operation.calibrating.*
status.operation.instrument.*
status.operation.instrument.*
status.operation.measuring.*
status.questionable.*
status.questionable.calibration.*
status.questionable.instrument.*
status.questionable.instrument.smuX.*
status.questionable.over_temperature.*
status.questionable.unstable_output*
status.request_enable
status.request_event
status.reset
status.standard.*
status.system.*
status.system2.*
status.system3.*
status.system4.*
status.system5.*

timer functions, page 12-108

timer.measure.t
timer.reset

trigger functions, page 12-109

trigger.clear
trigger.wait

tsplink function and attributes, page 12-110

tsplink.group
tsplink.master
tsplink.node
tsplink.readbit
tsplink.readport
tsplink.reset
tsplink.state
tsplink.trigger[N].assert
tsplink.trigger[N].clear
tsplink.trigger[N].mode
tsplink.trigger[N].overrun
tsplink.trigger[N].pulsewidth
tsplink.trigger[N].release
tsplink.trigger[N].wait
tsplink.writebit
tsplink.writeport
tsplink.writeprotect

userstring functions, page 12-116

userstring.add
userstring.catalog
userstring.delete
userstring.get

waitcomplete function, page 12-118

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.

beeper.beep	
Function	Generates a beep tone.
Usage	`beeper.beep(duration, frequency)` `duration` Set from 0.1 to 100 (seconds). `frequency` Set to 453, 621, 987 or 2400 (Hz).
Remarks	• There are four beeper frequencies: 453Hz, 621Hz, 987Hz and 2400Hz. If you set `frequency` to a different value, the closest supported frequency will be selected. • The beeper will not sound if it is disabled (see beeper.enable attribute). • 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 waitcomplete function can be used to hold up script execution until the beep command finishes.
Also see	beeper.enable

Example	Enables the beeper and generates a two-second, 2400Hz beep:
	`beeper.enable = 1`
	`beeper.beep(2, 2400)`

beeper.enable	
Attribute	Beeper control (on/off).
Usage	`beeperstate = beeper.enable` Reads beeper state. beeper.enable = beeperstate Writes beeper state. Set `beeperstate` to one of the following values: `0` Beeper disabled `1` Beeper enabled
Remarks	• This attribute enables or disables the beeper. Disabling the beeper also disables front panel key clicks. • Cycling power enables the beeper. The `reset` function does not affect the beeper state.
Also see	beeper.beep

Example	Enables the beeper and generates a two second, 2400Hz beep:
	`beeper.enable = 1`
	`beeper.beep(2, 2400)`

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.

bit.bitand

Function	Performs a bitwise logical AND operation on two numbers.
Usage	`value = bit.bitand(value1, value2)` `value1` First number for the AND operation. `value2` Second number for the AND operation. `value` Returned result of the AND operation.
Remarks	• This function performs a logical AND operation on two numbers. • Any fractional parts of `value1` and `value2` are truncated to make them integers. The returned `value` is also an integer. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.bitor, bit.bitxor

Example	AND'ing decimal 10 (binary 1010) with decimal 9 (binary 1001) will return a value of decimal 8 (binary 1000):
	`value = bit.bitand(10, 9)`
	`print(value)`
	Output: `8.000000e+00`

bit.bitor

Function	Performs a bitwise logical OR operation on two numbers.
Usage	`value = bit.bitor(value1, value2)` `value1` First number for the OR operation. `value2` Second number for the OR operation. `value` Returned result of the OR operation.

Remarks	• This function performs a logical OR operation on two numbers. • Any fractional parts of `value1` and `value2` are truncated to make them integers. The returned `value` is also an integer. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.bitand, bit.bitxor

Example	OR'ing decimal 10 (binary 1010) with decimal 9 (binary 1001) will return a `value` of decimal 11 (binary 1011):
	`value = bit.bitor(10, 9)`
	`print(value)`
	Output: `1.100000e+01`

bit.bitxor

Function	Performs a bitwise logical XOR (Exclusive OR) operation on two numbers.
Usage	`value = bit.xor(value1, value2)` `value1` First number for the XOR operation. `value2` Second number for the XOR operation. `value` Returned result of the XOR operation.
Remarks	• This function performs a logical Exclusive OR operation on two numbers. • Any fractional parts of `value1` and `value2` are truncated to make them integers. The returned `value` is also an integer. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.bitand, bit.bitor

Example	XOR'ing decimal 10 (binary 1010) with decimal 9 (binary 1001) will return a `value` of decimal 3 (binary 0011):
	`value = bit.bitxor(10, 9)`
	`print(value)`
	Output: `3.000000e+00`

bit.clear

Function	Clears a bit at a given index position.
Usage	`value = bit.clear(value1, index)` `value1` Given number. index Index position of the bit to be cleared (1 to 32). `value` Returns the result of the manipulation.
Remarks	• This function clears a bit at a given index position. • Any fractional part of `value1` is truncated to make it an integer. The returned `value` is also an integer. • The least significant bit of the given number is at index 1. The most significant bit is at index 32. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.get, bit.getfield, bit.set, bit.setfield, bit.test, bit.toggle

Example	The binary equivalent of decimal 15 is 1111. If you clear the bit at index position 2, the returned decimal `value` would be 13 (binary 1101):
	`value = bit.clear(15, 2)`
	`print(value)`
	Output: `1.300000e+01`

bit.get

Function	Retrieves the weighted value of a bit at a given index position.

Usage	`value = bit.get(value1, index)` `value1` Given number. `index` Index position of the bit to be retrieved (1 to 32). `value` Returned weighted value of the bit.
Remarks	• This function returns the value of the bit in `value1` at the given index. This is the same as returning `value1` with all other non-indexed bits set to zero. • 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. • If the indexed bit for the number is set to 0, the result will be 0. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.clear, bit.getfield, bit.set, bit.setfield, bit.test, bit.toggle

Example	The binary equivalent of decimal 10 is 1010. Getting the bit at index position 4 will return decimal `value` 8:
	`value = bit.get(10, 4)`
	`print(value)`
	Output: `8.000000e+00`

bit.getfield

Function	Returns a field of bits starting at a given index position.
Usage	`value = bit.getfield(value1, index, width)` `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.
Remarks	• 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`. • Prior to retrieving the field of bits, 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. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.clear, bit.get, bit.set, bit.setfield, bit.test, bit.toggle

Example	The binary equivalent of decimal 13 is 1101. The field at `index2` and `width3` consists of the binary bits 110. The returned `value` will be decimal 6 (binary 110):
	`value = bit.getfield(13, 2, 3)`
	`print(value)`
	Output: `6.000000e+00`

bit.set

Function	Sets a bit at a given index position.
Usage	`value = bit.set(value1, index)` `value1` Given number. `index` Index position of the bit to be set (1 to 32). `value` Returned value of the new number.
Remarks	• This function returns `value`, which is `value1` with the indexed bit set. The `index` 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. • Any fractional part of `value1` will be truncated to make it an integer. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.clear, bit.get, bit.getfield, bit.setfield, bit.test, bit.toggle

Example	The binary equivalent of decimal 8 is 1000. If the bit at `index3` is set to 1, the returned `value` will be decimal 12 (binary 1100):
	`value = bit.set(8, 3)`
	`print(value)`
	Output: `1.200000e+01`

bit.setfield

Function	Overwrites a bit field at a given index position.
Usage	`value = bit.setfield(value1, index, width, fieldvalue)` `value1` Given number. `index` Index position of the least significant bit of the field; 1 to (33 – `width`). `width` Field width – number of bits in the field; 1 to 24. `fieldvalue` Value to write to the field. `value` Returned value of the new number.
Remarks	• This function returns `value`, which is `value1` with a field of bits overwritten, starting at the given `index` position. The `index` specifies the position of the least significant bit of the given field. The `width` bits starting at the given index will be set to the value given by `fieldvalue`. The least significant bit in `value1` has an index of 1 and the most significant bit has an index of 32. • Prior to setting the field of bits, any fractional parts of `value1` and `fieldvalue` will be truncated to make them integers. • If the `fieldvalue` is wider than the `width`, the extra most significant bits of the `fieldvalue` will be truncated. For example, assume the `width` is 4 bits, and the binary value for `fieldwidth` is 11110 (5 bits). The most significant bit of `fieldwidth` will be truncated, and a binary value of 1110 will be used as the `fieldvalue`. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.clear, bit.get, bit.getfield, bit.set, bit.test, bit.toggle

Example	The binary equivalent of decimal 15 is 1111. After overwriting it with a decimal 5 (binary 101) at index position 2, the returned `value` will be decimal 11 (binary 1011):
	`value = bit.setfield(15, 2, 3, 5)`
	`print(value)`
	Output: `1.100000e+01`

bit.test	
Function	Returns the Boolean value (true or false) of a bit at a given index position.
Usage	`value = bit.test(value1, index)` `value1` Given number. `index` Index position of the bit to be tested (1 to 32). value Returned decimal value of the bit.
Remarks	• This function returns `value`, 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. • Any fractional part of `value1` will be truncated to make it an integer. If the indexed bit for `value1` is set to 0, the returned value will be false. If the indexed bit for `value1` is set to 1, the returned value will be true. • If the index is bigger than the number of bits in `value1`, the result will be false. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.clear, bit.get, bit.getfield, bit.set, bit.setfield, bit.toggle

Example	The binary equivalent of decimal 10 is 1010. Testing the bit at index position 4 will return a Boolean `value` of true:
	`value = bit.test(10, 4)`
	`print(value)`
	Output: `true`

bit.toggle	
Function	Toggles the value of a bit at a given index position.
Usage	`value = bit.toggle(value1, index)` `value1` Given number. `index` Index position of the bit to be toggled (1 to 32). value Returned value of the new number.
Remarks	• This function returns `value`, which is the result of toggling a bit in `value1`. • Any fractional part of `value1` 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. • The indexed bit for `value1` is toggled from 0 to 1, or 1 to 0. • See "Logic and bit operations" on page 12-10 for more information.
Also see	bit.clear, bit.get, bit.getfield, bit.set, bit.setfield, bit.test

Example	The binary equivalent of decimal 10 is 1010. Toggling the bit at index position 3 will return a decimal `value` of 14 (binary 1110).
	`value = bit.toggle(10, 3)`
	`print(value)`
	Output: `1.400000e+01`

data queue functions

You can use the data queue commands to share data between test scripts running in parallel and to access data from a remote group or a local node on a TSP-Link network. You can access data from the data queue even if a remote group or a local node has overlapped operations in process.

dataqueue.add	
Function	Adds an entry into the data queue.

Usage	`results = dataqueue.add (value, [timeout] )` value The data item to add. timeout The maximum number of seconds to wait for room in the data queue. results Assigns the value `true` or `false` based on the success of the add function. Replace the word results with the name of the variable that you want to store as the result indicator.
Remarks	• You can only use the `timeout` value while adding data from a local data queue. • The `timeout` value is ignored if the data queue is not full. • The `dataqueue.add` function returns false if time-out expires before room is available in the data queue or if the data queue is full and a `timeout` value is not specified. • If the value is a table, a duplicate of the table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example	• `dataqueue.add (10)`
	• `dataqueue.add(10, 2)`
	• `data_added = dataqueue.add (10, 3)`
	
	`Use the following code to verify data was added to the data queue.`
	`if not data_added then`
	`print ("timeout error")`
	`end`

dataqueue.CAPACITY

Attribute	The maximum number of entries that you can store in the data queue.
Usage	`capacity = dataqueue.CAPACITY` capacity A custom variable that stores the maximum number of entries in the data queue.
Remarks	A read only attribute.
Example	`print(dataqueue.CAPACITY)`

dataqueue.clear

Function	Clears the data queue.
Usage	`dataqueue.clear()`
Remarks	• The `dataqueue.clear` command forces all dataqueue.add commands in progress to time-out. • The function deletes all data from the data queue.

dataqueue.count

Attribute	Stores the number of entries saved in the data queue.
Usage	`count = dataqueue.count` count A custom variable that stores the number of entries in the data queue.
Remarks	This is a read-only attribute.

dataqueue.next

Function	Removes the next entry from the data queue.
Usage	`value = dataqueue.next ( [timeout] )` timeout The maximum number of seconds to wait for data in the data queue. value The next entry in the data queue.
Remarks	• If the data queue is empty, the function waits up to the `timeout` value. • If data is not available in the data queue before the timeout value expires, the return value is `nil`. • The entries in the data queue are removed in a first in and first out order. • If the value is a table, a duplicate of the original table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

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.

delay	
Function	Delays system operation.
Usage	`delay(seconds)` `seconds` Set the delay in seconds (100000 seconds maximum).
Remarks	• This function will delay for the specified number of seconds. It is impossible to delay for zero seconds. • Delays smaller than 50µs will be dominated by overhead such that the actual delay might be as long as 50µs (typical). For delays longer than 50µs, the delay may be as much as 10µs (typical) more than the requested delay.

Example	-- Sets SMU A output to 1V, soaks the DUT for 50ms and then turns the output off:
	`smua.source.levelv = 1.0`
	`delay(0.050)`
	`smua.source.off()`

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.

digio.readbit	
Function	Reads one digital I/O line.
Usage	`data = digio.readbit(N)` data A custom variable that stores the state of the I/O line. N Digital I/O number to be read (1 - 14).

Remarks	A returned value of 0 indicates that the line is low. A returned value of 1 indicates that the line is high.
Details	See "Digital I/O port" in Section 10.
Also see	digio.readport, digio.writebit, digio.writeport

Example	-- Assume line 4 is set high, and it is then read:
	`data = digio.readbit(4)`
	`print(data)`
	Output: `1.000000e+00`

digio.readport	
Function	Reads the digital I/O port.
Usage	`data = digio.readport()`
Remarks	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 00000010101010. Lines 2, 4, 6 and 8 are high (1), and the other 10 lines are low (0).
Details	See "Digital I/O port" in Section 10.
Also see	digio.readbit, digio.writebit, digio.writeport

Example	-- Assume lines 2, 4, 6 and 8 are set high, and the I/O port is then read:
	`data = digio.readport()`
	`print(data)`
	Output: `1.700000e+02` (binary 10101010)

digio.trigger[N].assert	Replace N with the number of the digital I/O trigger line: 1 to 14.
Function	Asserts a trigger on one of the digital I/O lines.
Usage	`digio.trigger[N].assert()`
Remarks	The set pulse width determines how long the trigger is asserted.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	digio.trigger[N].pulsewidth
Example	-- Asserts a trigger on I/O line 2: `digio.trigger[2].assert()`

digio.trigger[N].clear	Replace N with the number of the digital I/O trigger line: 1 to 14.
Function	Clears a trigger event on a digital I/O line.
Usage	`digio.trigger[N].clear()`
Remarks	• The trigger event detection recalls if a trigger event has been detected since the last digio.trigger[line].wait call. • This function clears a trigger event detector, discards the previous history of the trigger line and clears the digio.trigger[N].overrun attribute.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	digio.trigger[N].wait
Example	`-- Clears trigger event on I/O line 2:` `digio.trigger[2].clear()`

digio.trigger[N].mode	Replace N with the number of the digital I/O trigger line: 1 to 14.
Attribute	The trigger operation and detection mode.
Usage	`trig_mode = digio.trigger[N].mode` digio.trigger [N].mode = mode N The trigger line number. trig_mode The active trigger mode. mode Selects the current trigger mode. Choose one the following values for mode: `0` or `digio.TRIG_BYPASS` Allows direct control of the line. `1` or `digio.TRIG_FALLING` Detects falling edge input triggers. Asserts TTL-low pulse as an output trigger. `2` or `digio.TRIG_RISING` If the programmed state of the line is high, the `digio.TRIG_RISING` mode behaves similar to `digio.TRIG_RISINGA`. If the programmed state of the line is low, the `digio.TRIG_RISING` mode behaves similar to `digio.TRIG_RISINGM`. `3` or `digio.TRIG_EITHER` Detects rising or falling edge triggers. Asserts a TTL-low trigger pulse. `4` or `digio.TRIG_SYNCHRONOUSA` Detects the falling edge input triggers and automatically latches and drives the trigger line low. Asserting the output trigger releases the latched line. `5` or `digio.TRIG_SYNCHRONOUS` Detects the falling edge input triggers and automatically latches and drive the trigger line low. Asserts a TTL-low pulse as an output trigger. `6` or `digio.TRIG_SYNCHRONOUSM` Detects rising edge triggers as an input. Asserts a low TTL-low pulse for output. `7` or `digio.TRIG_RISINGA` Detects Rising Edge triggers as an input. Asserts a low TTL-low pulse as an output. `8` or `digio.TRIG_RISINGM` Edge detection as an input is not available. Generates a TTL-high pulse as an output trigger.
Remarks	• You can express the mode as a number (0 through 8) or you can use one of the pre-defined constants (see "Usage"). • The custom variable mode stores the trigger mode as a numeric value when the attribute is read. • The default trigger mode for a line is `digio.TRIG_BYPASS`. • To control the line state, use the `digio.TRIG_BYPASS` mode with the `digio.writebit` and the `digio.writeport` commands. • (Firmware version prior to 1.4.0 only) Use the `digio.TRIG_SYNCHRONOUS` mode for backward firmware compatibility. • (Firmware version 1.4.0 and higher) Use `digio.TRIG_SYNCHRONOUSA` or `digio.TRIG_SYNCHRONOUSM` modes. • (Firmware version prior to 1.4.0 only) Use the `digio.TRIG_RISING` mode for backward compatibility. • (Firmware version 1.4.0 and higher) Use `digio.TRIG_RISINGA` or `digio.TRIG_RISINGM` modes.
Details	See "Controlling digital I/O lines" in Section 10, digio.writebit, and digio.writeport.
Example	-- Sets the trigger mode for the I/O line 7 to digio.TRIG_RISINGM. `digio.trigger [7] = 8`

digio.trigger[N].overrun	Replace N with the number of the digital I/O trigger line: 1 to 14
Attribute	Use this attribute to read the trigger overrun status.
Usage	`overrun = digio.trigger[N].overrun` overrun The trigger overrun state.
Remarks	• A read only attribute. • Indicates an event was ignored because the event detector was in the detect state when the event was detected.

digio.trigger[N].pulsewidth	Replace N with the number of the digital I/O trigger line: 1 to 14.
Attribute	The length of time that the trigger line will be asserted for output triggers.
Usage	`width = digio.trigger[N].pulsewidth` Reads pulse width. `digio.trigger[N].pulsewidth = width` Writes pulse width. `width` The pulse width length (seconds).
Remarks	• Setting pulsewidth to 0 (seconds) asserts the trigger indefinitely. • The default pulsewidth time is 10µs.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	digio.trigger[N].release
Example	-- Sets the pulse width for trigger line 4 to 20µs: `digio.trigger[4].pulsewidth = 20e-6`

digio.trigger[N].release	Replace N with the number of the digital I/O trigger line: 1 to 14.
Function	Releases an indefinite length or latched trigger.
Usage	`digio.trigger[N].release()`
Remarks	Releases a trigger that was asserted with an indefinite pulse width, as well as a trigger that was latched in response to receiving a synchronous mode trigger.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	digio.trigger[N].pulsewidth
Example	Releases trigger line 4: `digio.trigger[4].release()`

digio.trigger[N].wait	Replace N with the number of the digital I/O trigger line: 1 to 14.
Function	Waits for a trigger.
Usage	`triggered = digio.trigger[N].wait(timeout)` timeout Specifies the time-out value in seconds. triggered A customized variable that stores the value true if a trigger is detected, or false if a trigger is not detected during the time-out period.
Remarks	This function waits up to the `timeout` value in seconds for an input trigger. If one or more trigger events are detected since the last time `digio.trigger[N].wait` or `digio.trigger[N].clear` was called, this function immediately returns. After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	digio.trigger[N].clear

Example	Waits up to three seconds for a trigger to be detected on trigger line 4, then displays if the trigger was detected: `triggered = digio.trigger[4].wait(3)` print(triggered) Output: `false` Triggers are not detected. `true` Triggers are detected.

digio.writebit	
Function	Sets a digital I/O line high or low.
Usage	`digio.writebit(N, data)` `N` The digital I/O line number (1 to 14). `data` The value to write to the bit; 0 (low) or 1 (high).
Remarks	• If the output line is write protected, via the digio.writeprotect attribute, the command will be ignored. • The `reset` function does not affect the present state of the digital I/O lines. • Use the `digio.writebit` and `digio.writeport` commands to control the output state of the synchronization line when the trigger mode is set to `digio.TRIG_BYPASS`.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	digio.readbit, digio.readport, digio.trigger[N].mode
Example	-- Sets digital I/O line 4 low (0): `digio.writebit(4, 0)`

digio.writeport	
Function	Writes to all digital I/O lines.
Usage	`digio.writeport(data)` `data` Value to write to the port; 0 to 16383.
Remarks	• The binary representation of data indicates the output pattern to be written to the I/O port. For example, a `data` value of 170 has a binary equivalent of 00000010101010. Lines 2, 4, 6 and 8 are set high (1), and the other 10 lines are set low (0). • Write protected lines will not be changed (see digio.writeprotect). • The `reset` function does not affect the present states of the digital I/O lines. • Use the `digio.writebit` and `digio.writeport` commands to control the output state of the synchronization line while the trigger mode is set to digio.TRIG_BYPASS.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	digio.readbit, digio.readport, digio.writebit, digio.writebit
Example	Sets digital I/O lines 1 through 8 high (binary 00000011111111): `digio.writeport(255)`

digio.writeprotect	
Attribute	Write protect mask that disables bits from being changed with the `digio.writebit` and `digio.writeport` functions.
Usage	`mask = digio.writeprotect` Reads write protect mask. `digio.writeprotect = mask` Writes write protect mask. `mask` Set to the value that specifies the bit pattern for write protect.
Remarks	• Bits that are set to 1 cause the corresponding line to be write protected. • The binary equivalent of `mask` indicates the mask to be set for the I/O port. For example, a `mask` value of 7 has a binary equivalent 00000000000111. This mask write protects lines 1, 2 and 3.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	digio.writebit, digio.writeport
Example	Write protects lines 1, 2, 3 and 4: `digio.writeprotect = 15`

display functions and attributes

The functions and attributes in this group are used for various display operations, which are explained in Section 14.

display.clear	
Function	Clears all lines of the display.
Usage	`display.clear()`
Remarks	• This function will switch to the user screen and then clear the display. • The `display.clear()`, `display.setcursor`, and `display.settext` 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.
Details	See "Clearing the display" in Section 14.
Also see	display.setcursor, display.settext

display.getannunciators

Function	Reads the annunciators that are presently turned on.
Usage	`annun = display.getannunciators()` annun Returns the bitmap value for annunciators that are turned on.
Remarks	This function returns a bitmap 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: 0000010000000100 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 (**4W** and **REM**). Notice that the sum of the weighted values for bits 3 and 11 is the returned value (1028). AnnunciatorBit Weighted Value Annunciator Bit Weighted Value FILT1 1 EDIT 9256 MATH 2 2 ERR 10512 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
Details	See "Annunciators" in Section 14.

Example	Reads the annunciators that are turned on:
	`annun = display.getannunciators()`
	`print(annun)`
	Output: `1.280000e+03`
	For the returned value of 1280, the binary equivalent is 0000010100000000. Bits 9 and 11 are set. Using the above chart in "**Remarks**", the **REM** and **EDIT** annunciators are turned on.

display.getcursor

Function	Reads the present position of the cursor for the user display.
Usage	`row, column, style = display.getcursor()` row Returns the row for the present cursor position. column Returns the column for the present cursor position. style Returns the cursor style.
Remarks	• This function switches the display to the user screen, and then returns values to indicate row and column position, and cursor style. • The `row` value is returned as 1 (top row) or 2 (bottom row). • With the cursor in the top row, the `column` is returned as a value from 1 to 20. With the cursor in the bottom row, the `column` is returned as a value from 1 to 32. Columns are numbered from left to right on the display. • The returned value for style is 0 (invisible) or 1 (blink).
Details	See "Cursor position" in Section 14.
Also see	display.gettext, display.screen, display.setcursor, display.settext

Example	Reads cursor position (row and column):
	`row, column = display.getcursor()`
	`print(row, column)`
	Output: `1.000000e+00 3.000000e+00`
	The above output indicates that the cursor is in Row 1 at Column 3.

display.getlastkey

Function	Retrieves the keycode for the last pressed key.
Usage	`key = display.getlastkey()`

Remarks	• This read-only function returns the keycode for the last pressed key. `key` returns one of the following values:
	`0 (display.KEY_NONE)82 (display.KEY_ENTER)`
	`65 (display.KEY_RANGEUP)83 (display.KEY_MEASB)`
	`67 (display.KEY_RELB)84 (display.KEY_DIGITSB)`
	`68 (display.KEY_MENU)85 (display.KEY_RECALL)`
	`69 (display.KEY_MODEA)86 (display.KEY_MEASA)`
	`70 (display.KEY_RELA)87 (display.KEY_DIGITSA)`
	`71 (display.KEY_RUN)90 (display.KEY_LIMITB)`
	`72 (display.KEY_DISPLAY)91 (display.KEY_SPEEDB)`
	`73 (display.KEY_AUTO)92 (display.KEY_TRIG)`
	`74 (display.KEY_FILTERB)93 (display.KEY_LIMITA)`
	`75 (display.KEY_EXIT)94 (display.KEY_SPEEDA)`
	`76 (display.KEY_SRCB)95 (display.KEY_LOAD)`
	`77 (display.KEY_FILTERA)97 (display.WHEEL_ENTER)`
	`78 (display.KEY_STORE)103 (display.KEY_RIGHT)`
	`79 (display.KEY_SRCA)104 (display.KEY_LEFT)`
	`80 (display.KEY_CONFIG)114 (display.WHEEL_RIGHT)`
	`81 (display.KEY_RANGEDOWN)`
	• A history of the keycode for the last pressed front panel key is maintained by the Series 2600. When the instrument is powered-on, (or when transitioning from local to remote), the keycode is set to `0` (`display.KEY_NONE`).
	• The **OUTPUT ON/OFF** keys for SMU A and SMU B cannot be tracked by this function.
	• Pressing the **EXIT**/**LOCAL** key normally aborts a script. In order to use this function with the **EXIT** key, `display.locallockout` must be used.

Details	See "Sending keycodes" in Section 14.
Also see	display.sendkey, display.locallockout

Example	On the front panel, press the **MENU** key and then send the following code:
	`key = display.getlastkey()`
	`print(key)`
	Output: `6.800000e+01`

display.gettext

Function	Reads the text presently displayed.

Usage	There are five ways to use this function:
	`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 all character codes.
	`row` Set to `1` or `2` to select which row to read text. If not included, text from both rows are read.
	`column_start` Set to starting column for reading text. Default is `1`.
	`column_end` Set to ending column for reading text. Default is `20` (Row 1) or `32` (Row 2).
	
	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.

Remarks	• Sending the command without any parameters returns both lines of the display. The `$N` character code will be included to show where the top line ends and the bottom line begins. • With `embellished` set to `true`, all other character codes will be returned along with the message. With `embellished` set to false, only the message and the `$N` character code will be returned. See the display.settext function for details on the character codes. • The display will not be switched to the user screen. Text will be read from the active screen.
Details	See "Displaying text messages" in Section 14.
Also see	display.getcursor, display.setcursor, display.settext

Example	Returns all text in both lines of the display:
	`text = display.gettext()`
	`print(text)`
	Output: `User Screen $N`
	The above output indicates that the message "User Screen" is on the top line. The bottom line is blank.

display.inputvalue

Function	Displays a formatted input field that the operator can edit.

Usage	There are four ways to use this function:
	`value = display.inputvalue(format)`
	`value = display.inputvalue(format, default)`
	`value = display.inputvalue(format, default, min)`
	`value = display.inputvalue(format, default, min, max)`
	format — Define format string for the input field using 0, the decimal point (.), polarity sign (+) and 'E' for exponent.
	default — Set the default value for the parameter.
	min — Set the minimum input value that can be set.
	max — Set the maximum input value that can be set.

Remarks	• 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. • Examples of the input field: +0.0000+00.0000E+000.00000E+0 • **Value field**: + Include a "+" sign for positive/negative value entry. Not including the "+" sign prevents negative value entry. 0 Defines the digit positions for the value. Up to six 0 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 Defines the digit positions for the exponent. • Along with specifying the `format` 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). • 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 `default` value. • 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. • The input value is limited to ±1e37.
Remarks (cont.)	• After sending this command, script execution waits for the operator to enter a value and press **ENTER**: • If limits are used, the operator will not be able to input values outside the minimum and maximum limits. • 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. • After sending this command and pressing the **EXIT** key, `value` will return `nil`.
Details	See "Parameter value prompting" in Section 14.
Also see	display.prompt, display.setcursor, display.settext

Example	Displays an editable field ("+0.50") for operator input – Valid input range is -0.10 to +2.00, with a default of 0.50:
	`display.clear()`
	`value = display.inputvalue("+0.00", 0.5, -0.1, 2.0)`

display.loadmenu.add

Function	Adds an entry to the “USER TESTS” submenu of the “LOAD TEST” menu.

Usage	There are two ways to use this function:
	`display.loadmenu.add(displayname, chunk)`
	`display.loadmenu.add(displayname, chunk, memory)`
	`displayname` Name to display in the menu.
	`chunk` Chunk is the code to be executed.
	`memory` Save or don’t save `chunk` and `displayname` 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`.

Remarks	• 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 `chunk` will be executed when the **RUN** key is pressed. • 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” in Section 14). If the script is not saved in non-volatile memory, it will be lost when the Series 2600 is turned off. See **Example 1** below. • 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.
Details	See “Load test menu” in Section 14.
Also see	display.loadmenu.delete

Examples	**Example 1** – Assume a script with a function named “DUT1” has already been loaded into the Series 2600, 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 Series 2600. 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 – 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:
	`display.loadmenu.add("Part1", "testpart([[Part1]], 5.0)", display.SAVE)`

display.loadmenu.delete

Function	Deletes an entry from the “USER” submenu of the “LOAD TEST” menu.
Usage	`display.loadmenu.delete(displayname)` `displayname` Name to remove from the menu.
Remarks	This function is used to delete an entry (displayname) from the front panel USER TESTS submenu of the LOAD TEST menu.
Details	See “Load test menu” in Section 14.
Also see	display.loadmenu.add
Example	Removes the entry named “Part1” from the front panel “USER TESTS” load menu: `display.loadmenu.delete("Part1")`

display.locallockout

Attribute	**LOCAL** key disabled.
Usage	`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 **LOCAL** key. `1` or `display.LOCK` Locks out **LOCAL** key.
Remarks	Setting `display.locallockout` to `display.LOCK` prevents the user from interrupting remote operation by pressing the **LOCAL** key. Set this attribute to `display.UNLOCK` to allow the **LOCAL** key to abort script/remote operation.
Details	See "LOCAL lockout" in Section 14.
Example	Disables the front panel **LOCAL** key: `display.locallockout = display.LOCK`

display.menu

Function	Presents a menu on the front panel display.
Usage	`selection = display.menu(name, items)` `name` Menu name to display on the top line. `items` Menu items to display on the bottom line.
Remarks	• The menu consists of the menu `name` string on the top line, and a selectable list of `items` on the bottom line. The menu items must be a single string with each item separated by white space. The `name` for the top line is limited to 20 characters. • After sending this command, script execution 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 **ENTER** key). When an item is selected, the text of that selection is returned. • 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.
Details	See "Menu" in Section 14.
Example	Displays a menu with three menu items. If the second menu item is selected, `selection` will be given the value `Test2`: `selection = display.menu("Sample Menu", "Test1 Test2 Test3")` print(selection) Output: Test2

display.numpad

Attribute	This attribute controls whether the front panel keys act as a numeric keypad during value entry.
Usage	`X = display.numpad` Read the numpad option. `display.numpad = X` Write the numpad option. where X is one of: 1 or `display.ENABLE` Enable the numeric keypad feature. 0 or `display.DISABLE` Disable the numeric keypad feature.
Remarks	The numeric keypad feature is only available when editing a numeric value and the EDIT annunciator is lit.
Example	Turn on the numeric keypad feature: `display.numpad = 1`

display.prompt

Function	Prompts the user to enter a parameter from the front panel.

Usage	There are four ways to use this function:
	`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)`
	format — Define format string for the input field using 0, the decimal point (.), polarity sign (+) and 'E' for exponent.
	units — Set units text string for top line (8 characters maximum).
	help — Text string to display on the bottom line (32 characters maximum).
	default — Set the default value for the parameter.
	min — Set the minimum input value that can be set.
	max — Set the maximum input value that can be set.

Remarks	• 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 • The `format` configures the editable input field. Four examples for the format: +0.0000+00.0000E+000.00000E+0 • **Value field**: • + Include a "+" sign for positive/negative value entry. Not including the "+" sign prevents negative value entry. • 0 Defines the digit positions for the value. Up to six 0 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)**: • EInclude the "E" for exponent entry. • + Include a "+" sign for positive/negative exponent entry. Not including the "+" sign prevents negative exponent entry. • 0 Defines the digit positions for the exponent. • `units` is a string that indicates the units (e.g., "V" or "A") for the value and `help` provides a message prompt on the bottom line. • Along with specifying the `format` 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). • 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 `default` value. • 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. • The input value is limited to ±1e37. • After sending this command, script execution holds and waits for the operator to enter a value and press **ENTER**: • If limits are used, the operator will not be able to input values outside the minimum and maximum limits. • 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. • After sending this command and pressing the **EXIT** key, `value` will return `nil`.
Details	See "Parameter value prompting" in Section 14.
Also see	display.inputvalue

Example	Prompts the operator to enter a voltage value – Valid input range is 0 to +2.00, with a default of 0.50: `value = display.prompt("0.00", "V", "Input 0 to +2V" 0.5, 0, 2)` The above command will display the following input prompt: 0.50V Input 0 to +2V

display.screen

Attribute	The selected display screen.
Usage	`displayid = display.screen` -- Reads display screen. `display.screen = displayid` -- Writes display screen. Set displayid to one of the following values: 0 or display.SMUA — Displays source-measure and compliance for SMU A. 1 or display.SMUB — Displays source-measure and compliance for SMU B. 2 or display.SMUA_SMUB — Displays source-measure for SMU A and SMU B. 3 or display.USER — Displays the user screen.
Remarks	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.
Details	See "Display screen" in Section 14.
Example	Selects the source-measure and compliance limit display for SMUA: `display.screen = display.SMUA`

display.sendkey

Function	Sends a keycode to simulate the action of a front panel control.

Usage	`display.sendkey(keycode)`
	Set keycode to one of the values shown below:
	`display.KEY_AUTO` or `73display.KEY_OUTPUTA` or `88`
	`display.KEY_CONFIG` or `80display.KEY_OUTPUTB` or `96`
	`display.KEY_DIGITSA` or `87display.KEY_RANGEDOWN` or `81`
	`display.KEY_DIGITSB` or `84display.KEY_RANGEUP` or `65`
	`display.KEY_DISPLAY` or `72display.KEY_RECALL` or `85`
	`display.KEY_ENTER` or `82display.KEY_RELA` or `70`
	`display.KEY_EXIT` or `75display.KEY_RELB` or `67`
	`display.KEY_FILTERA` or `77display.KEY_RIGHT` or `103`
	`display.KEY_FILTERB` or `74display.KEY_RUN` or `71`
	`display.KEY_LEFT` or `104display.KEY_SPEEDA` or `94`
	`display.KEY_LIMITA` or `93display.KEY_SPEEDB` or `91`
	`display.KEY_LIMITB` or `90display.KEY_SRCA` or `79`
	`display.KEY_LOAD` or `95display.KEY_SRCB` or `76`
	`display.KEY_MEASA` or `86display.KEY_STORE` or `78`
	`display.KEY_MEASB` or `83display.KEY_TRIG` or `92`
	`display.KEY_MENU` or `68display.WHEEL_ENTER` or `97`
	`display.KEY_MODEA` or `69display.WHEEL_LEFT` or `107`
	display.KEY_MODEBor66display.WHEEL_RIGHTor114

Remarks	Sending this command simulates the pressing of a front panel key or wheel, or turning the Wheel one click to the left or right.
Details	See "Sending keycodes" in Section 14.
Example	"Press" the **RUN** key: `display.sendkey(display.KEY_RUN)`

display.setcursor

Function	Sets the position of the cursor.

Usage	There are two ways to use this function:
	`display.setcursor(row, column)`
	`display.setcursor(row, column, style)`
	`row` Set `row` number for the cursor (1 or 2).
	`column` 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.
	`style` Set cursor `style` to be invisible (0) or blink (1).

Remarks	• Sending this command selects the user screen and then moves the cursor to the given location. • An out of range parameter for `row` will set the cursor to row 2. An out of range parameter for `column` will set the cursor to column 20 (for row 1) or 32 (for row 2). • An out of range parameter for `style` sets it to 0 (invisible). • A blinking cursor will only be visible when it is positioned over displayed text. It cannot be seen when positioned over a space character. • The `display.clear`, `display.setcursor`, and `display.settext` 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.
Details	See "Sending keycodes" in Section 14.
Also see	display.clear, display.getcursor, display.gettext, display.settext
Example	Positions cursor on row 2 column 1: `display.setcursor(2, 1)`

display.settext

Function	Displays text on the user screen.
Usage	`display.settext(text)` `text` Text message string to be displayed.
Remarks	• 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. • 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. • 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. • The text remains on the display until replaced or cleared. • The following character codes can be also be included in the `text` string: $NNewline – Starts text on the next line. If the cursor is already on line 2, text will be ignored after the '$N' is received. $RSets text to Normal. $B Sets text to Blink. $DSets text to Dim intensity. $FSets text to background blink. $$Escape sequence to display a single "$". • The `display.clear`, `display.setcursor`, and `display.settext` 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.
Details	See "Displaying text messages" in Section 14.
Also see	display.clear, display.getcursor, display.gettext, display.setcursor

Example	Displays a message on the user screen:
	`display.clear()`
	`display.settext("Message Test $N$Bwith Row 2 Blinking")`
	The top line displays “Message Test” and the bottom line displays the blinking message “with Row 2 Blinking”.

display.smuX.digits

X = SMU channel (a or b)

Attribute	The selected measurement display resolution.
Usage	`digits = display.smuX.digits` -- Reads resolution. display.smuX.digits = digits -- Writes resolution. Set `digits` to one of the following values: `4` or display.DIGITS_4_5 Selects 4-1/2d digit resolution. `5` or display.DIGITS_5_5 Selects 5-1/2d digit resolution. `6` or display.DIGITS_6_5 Selects 6-1/2d digit resolution.
Remarks	• This attribute selects the measurement display resolution; 4-1/2 digit, 5-1/2 digit or 6-1/2 digit. • SMU A and SMU B can be set for a different measurement display resolution.
Details	See “Display resolution” in Section 14.
Example	Selects 5-1/2d digit resolution for SMU A: `display.smua.digits = display.DIGITS_5_5`

display.smuX.measure.func

X = SMU channel (a or b)

Attribute	The type of measurement being displayed.
Usage	`func = display.smuX.measure.func` -- Reads function. `display.smuX.measure.func` = func-- Writes function. Set `func` to one of the following values: `0` or display.MEASURE_DCAMPS Selects current measure function. `1` or display.MEASURE_DCVOLTS Selects volts measure function. `2` or display.MEASURE_OHMS Selects ohms measure function. `3` or display.MEASURE_WATTS Selects power measure function.
Remarks	• Selects the displayed measurement function: Amps, volts, ohms or watts. • SMU A and SMU B can be set for different measurement functions.
Details	See “Measurement functions” in Section 14.
Example	Selects the current measure function for SMU A: `display.smua.measure.func = display.MEASURE_DCAMPS`

display.trigger.clear

Function	Clears the front panel trigger event detector.
Usage	`display.trigger.clear()`
Remarks	The trigger event detector remembers if an event has been detected since the last `display.trigger.wait` call. This function clears the trigger’s event detector and discards the previous history of **TRIG** key presses.
Details	See “Display triggering” in Section 14.
Also see	display.trigger.wait

display.trigger.wait

Function	Waits for the **TRIG** key on the front panel to be pressed.

Usage	`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.
Remarks	• This function will wait for the **TRIG** 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. • 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. • Use the `display.trigger.clear` call to clear the trigger event detector.
Details	See "Display triggering" in Section 14.
Also see	display.trigger.clear
Example	Waits up to five seconds for the **TRIG** key to be pressed: `triggered = display.trigger.wait(5)` print(triggered) Output: true The above output indicates that the **TRIG** key was pressed (trigger detected) before the five second timeout expired.

display.waitkey

Function	Captures the keycode value for the next key press.
Usage	`key = display.waitkey()`
Remarks	• 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. • If the **EXIT** key is pressed while this function is waiting for a keypress, the script will not be aborted. • A typical use for this function is to prompt the user to press **EXIT** to abort the script or press any other key to continue. If keycode value 75 is returned (**EXIT** key pressed), then the `exit()` function can be called to abort the script. Sample code for this process is provided in "Capturing key-press codes" in Section 14. ControlKeycodeControlKeycodeControlKeycode AUTO 73LIMIT (B)90REL (A)70 CONFIG80LOAD95REL (B)67 CURSOR (left)104MEAS (A)86RUN71 CURSOR (right)103MEAS (B)83SPEED (A)94 DIGITS (A)87MENU68SPEED (B)91 DIGITS (B)84MODE (A)69SRC (A)79 DISPLAY72MODE (B)66SRC (B)76 ENTER82OUTPUT (A)88STORE78 EXIT75OUTPUT (B)96TRIG92 FILTER (A)77RANGE (down)81Wheel (press)97 FILTER (B)74RANGE (up)65Wheel (left)107 LIMIT (A)93RECALL85Wheel (right)114 The above chart lists the numeric keycode values for the front panel controls. The keycode value identifiers are listed in the documentation for display.sendkey (e.g., `display.KEY_RUN` is the identifier for the **RUN** key)
Details	See "Capturing key-press codes" in Section 14.
Also see	display.sendkey, display.settext, display.getlastkey

Example	The following code will hold up script execution and wait for the operator to press a key or the wheel, or rotate the wheel: `key = display.waitkey()` print(key) Output: `8.600000e+01` The above output (86) indicates that the **MEAS (A)** key was pressed.

errorqueue functions and attribute

The functions and attribute in this group are used to read the entries in the error/event queue.

errorqueue.clear	
Function	Clears all entries out of the error/event queue.
Usage	`errorqueue.clear()`
Remarks	This function removes all entries from the error/event queue.
Details	See Appendix B (error and status messages) and Appendix D (status model).
Also see	errorqueue.count, errorqueue.next

errorqueue.count	
Attribute	The number of entries in the error/event queue.
Usage	`count = errorqueue.count`
Remarks	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.
Details	See Appendix B (error and status messages) and Appendix D (status model).
Also see	errorqueue.clear, errorqueue.next

Example	Reads number of entries in the error/event queue:
	`count = errorqueue.count`
	`print(count)`
	Output: `4.000000e+00`
	The above output indicates that there are four entries in the event/error queue.

errorqueue.next	
Function	Reads an entry from the error/event queue.
Usage	`errorcode, message, severity, node = errorqueue.next()` errorcode — Returns the error code number for the entry. message — Returns the message that describes the entry. severity — Returns the severity level (0, 10, 20, 30 or 40). node — Returns the node number where the error originated.
Remarks	• Entries are stored in a first-in, first-out (FIFO) queue. This function reads the oldest entry and removes it from the queue. • Error codes and messages are listed in Table B-2 in Appendix B. • If there are no entries in the queue, code 0, "Queue Is Empty" is returned. • Returned `severity` levels include the following: 0 Informational – Indicates no error: "Queue is Empty". 10 Informational – Indicates an event or a minor error. Examples: "Reading Available" and "Reading Overflow". 20Recoverable – 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". 30 Serious – Indicates a serious error and may require technical assistance. Example: "Saved calibration constants corrupted". 30 Fatal – Indicates that the Series 2600 is non-operational and will require service. Contact information for service is provided in Section 1. Examples: "Bad SMU AFPGA image size", "SMU is unresponsive" and "Communication Timeout with DFPGA". • In an expanded system, each TSP-Link enabled instrument is assigned a node number. `node` returns the node number where the error originated.
Details	See Appendix B (error and status messages) and Appendix D (status model).
Also see	errorqueue.clear, errorqueue.count

Example	Reads the oldest entry in the error/event queue:
	`errorcode, message = errorqueue.next()`
	`print(errorcode, message)`
	Output: `0.000000e+00 Queue Is Empty`
	The above output indicates that the queue is empty.

exit function

This function is used to terminate a script that is presently running.

exit	
Function	Stops execution of a script.
Usage	`exit()`
Remarks	Terminates script execution when called from a script that is being executed. This command will not wait for overlapped commands to complete before terminating script execution. If overlapped commands are required to finish, use the `waitcomplete` function prior to calling `exit`.
Also see	"System behavior, Abort" in Section 9.

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

Attribute	The precision (number of digits) for all numbers printed with the ASCII format.
Usage	`precision = format.asciiprecision`-- Reads precision. `format.asciiprecision = precision`-- Writes precision. precision Set from 1 to 16.
Remarks	• This attribute selects the precision (number of digits) for data printed with the `print`, `printnumber` and `printbuffer` functions. The precision attribute is only used with the ASCII format. The precision must be a number between 1 and 16. • 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. • The default (`reset`) precision is 6.
Also see	format.byteorder, format.data, printbuffer, printnumber

Example	Sets the ASCII precision to 7 digits and prints a number:
	`format.asciiprecision = 7`
	`print(2.5)`
	Output: 2.500000E+00

format.byteorder

Attribute	The binary byte order for data printed using the `printnumber` and `printbuffer` functions.
Usage	`order = format.byteorder` -- Reads byte order. format.byteorder = order -- Writes byte order. Set `order` to one of the following values: `0` or format.NORMAL — Most significant byte first. `0` or format.BIGENDIAN — Most significant byte first. `0` or format.NETWORK — Most significant byte first. `1` or format.SWAPPED — Least significant byte first. `1` or format.LITTLEENDIAN — Least significant byte first.
Remarks	• This attribute selects the byte order that data is written when printing data values with the `printnumber` and the `printbuffer` functions. The byte order attribute is only used with the SREAL, REAL, REAL32, and REAL64 data formats. • NORMAL, BIGENDIAN, and NETWORK select the same byte order. SWAPPED and LITTLEENDIAN select the same byte order. They are alternative identifiers. Selecting which to use is a matter of preference. • Select the SWAPPED or LITTLEENDIAN byte order when sending data to an IBM PC compatible computer.
Also see	format.asciiprecision, format.data, printbuffer, printnumber
Example	Selects the SWAPPED byte order: `format.byteorder = format.SWAPPED`

format.data

Attribute	The data format for data printed using the `printnumber` and `printbuffer` functions.
Usage	`fmt = format.data` -- Reads data format. `format.data` = fmt -- Writes data format. Set `fmt` to one of the following values: `1` or format.ASCII — ASCII format. `2` or format.SREAL — Single precision IEEE-754 binary format. `2` or format.REAL32 — Single precision IEEE-754 binary format. `3` or format.REAL — Double precision IEEE-754 binary format. `3` or format.REAL64 — Double precision IEEE-754 binary format.
Remarks	• This attribute selects the data format used to print data values with the `printnumber` and `printbuffer` functions. • The precision of the ASCII format can be controlled with the format.asciiprecision attribute. The byte order of SREAL, REAL, REAL32, and REAL64 can be selected with the format.byteorder attribute. • REAL32 and SREAL select the same single precision format. REAL and REAL64 select the same double precision format. They are alternative identifiers. Selecting which to use is a matter of preference. • The IEEE-754 binary formats use 4 bytes each for single precision values and 8 bytes each for double precision values. • 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.
Also see	format.asciiprecision, format.byteorder, printbuffer, printnumber
Example	Selects the ASCII data format: `format.data = format.ASCII`

gpib attribute

The following attribute is used to set the GPIB address.

gpib.address	
Attribute	GPIB address.
Usage	`address = gpib.address` -- Reads address. `gpib.address = address` -- Writes address. address Set from 0 to 30.
Remarks	• 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. • 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. • The GPIB address is stored in non-volatile memory. The `reset` function has no effect on the address.
Details	See "GPIB operation" in Section 11.

Example	Sets the GPIB address of the Series 2600 to 26 and then reads the address:
	`gpib.address = 26`
	`address = gpib.address`
	`print(address)`
	Output: `2.600000e+01`

localnode attribute

Use the attributes and functions in this section to set the power line frequency, control prompts (on and off), control error messages, (show and hide), to access global variables, and to run test scripts.

There are two different ways to use the `localnode` object.

- Send commands from the local node
- Send commands from a remote node

To send commands from the local node, use the `localnode` element:

```
localnode.linefreq = 50
```

NOTE Use the node reference to send commands from a remote node, do not use the `localnode` element.

```
node [N].linefreq=50
```

localnode.autolinefreq	
Attribute	Automatic power line frequency detection control.
Usage	flag = localnode.autolinefreq -- Read auto line frequency detection setting. localnode.autolinefreq = true -- Enable automatic line frequency detection on power-up.
Remarks	• When this attribute is set to true, the power line frequency is detected automatically the next time the Series 2600 powers up. • After the power line frequency is automatically detected at power-up, the `localnode.linefreq` attribute will be set automatically to 50 or 60. • If the `localnode.linefreq` attribute is explicitly set, `localnode.autolinefreq` will be automatically set to false. • When used in an expanded system (TSP-Link), `localnode.autolinefreq` is sent to the remote master node only. • Use `node[N].autolinefreq` (where `[N]` is the node number) to send the command to any node in the system. See Section 9 for details on TSP-Link. • This command is compatible with firmware version 1.2.0 and later.
Also see	localnode.linefreq
Example	Disable automatic power line frequency detection: `localnode.autolinefreq = 0`

localnode.execute	
Function	Use this function to start test scripts on a remote node.
Usage	`node[N].execute (myscript)` myscript — Replace the custom variable myscript with the variable name or the source code.
Remarks	• Only the master node can issue the `execute` command to a remote node. • You cannot use the `execute` command to run test scripts on the master node.

Example	-- Runs script code stored in the custom variable sourcecode.
	`node[N].execute (sourcecode)`
	-- Runs script code in a string constant.
	`node[N].execute ("x=5")`
	-- Runs a test script that was loaded into memory.
	`node[N].execute (myscript.source)`

localnode.getglobal	
Function	This function returns the value of a global variable.
Usage	`value = node[N].getglobal(name)` name — The name of the global variable. value — The value of the global variable. N — The number of the remote node.
Remarks	• Use this function to retrieve the value of a global variable from a remote node. • Do not use this command to retrieve the value of a global variable from a local node.
Example	-- Retrieves and outputs the value of the global variable named meas_val from node 5. print(node[5].getglobal(“meas_val”))

localnode.linefreq

Attribute	Power line frequency.
Usage	`frequency = localnode.linefreq` Reads line frequency. `localnode.linefreq = frequency` Writes line frequency. frequency Set to 50 or 60.
Remarks	• 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. • When used in an expanded system (TSP-Link), `localnode.linefreq` is sent to the remote master node only. Use `node[N].linefreq` (where `N` is the node number) to send the command to any node in the system. See Section 9 for details on TSP-Link. • When this attribute is set, the `localnode.autolinefreq` attribute is automatically set to false.
Also see	localnode.autolinefreq
Example	Sets the line frequency to 60Hz: `localnode.linefreq = 60`

localnode.prompts

Attribute	Prompting mode.
Usage	`prompting = localnode.prompts`-- Reads prompting state. `localnode.prompts = prompting`-- Writes prompting state. prompting Set to 0 to disable or 1 to enable.
Remarks	• 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. • The command messages do not generate prompts. The Series 2600 generates prompts in response to command messages. • When the prompting mode is enabled, the Series 2600 generates prompts in response to command messages. There are three prompts that might be returned: •"TSP>" is the standard prompt. This prompt indicates that everything is normal and the command is done processing. •"TSP?" is issued if there are entries in the error queue when the prompt is issued. Like the "TSP>" prompt, it indicates the command is done processing. It does not mean the previous command generated an error, only that there are still errors in the queue when the command was done processing. •">>>>" is the continuation prompt. This prompt is used when downloading scripts 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. • Test Script Builder (TSB) requires prompts. It sets the prompting mode automatically. If you disable prompting, use of the TSB will freeze 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. • When used in an expanded system (TSP-Link), `localnode.prompt` is sent to the remote master node only. Use `node[N].prompt` (where `N` is the node number) to send the command to any node in the system. See Section 9 for details about TSP-Link.
Also see	`localnode.showerrors`
Example	`Enables prompting:` `localnode.prompts = 1`

localnode.setglobal	
Function	Sets the value of a global variable.
Usage	`node[N].setglobal (name, value)` `name` The name of the variable. `value` The value assigned to the variable.
Remarks	• Use this function to create a global variable and to assign values to the global variable from a remote master node. • Do not use this command to set the value of a global variable on the local node.
Example	`node[N].setglobal ("x", 5)`

localnode.showerrors	
Attribute	Automatic display of errors.
Usage	`errormode = localnode.showerrors`-- Reads show errors state. `localnode.showerrors = errormode`-- Writes show errors state. errormode - Set to 0 or 1.
Remarks	• If this attribute is set to 1, the unit will automatically display any generated 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). • If this attribute is set to 0, errors will be left in the error queue and must be explicitly read or cleared. • When used in an expanded system (TSP-Link), `localnode.showerrors` is sent to the remote master node only. Use `node[N].showerrors` (where `N` is the node number) to send the command to any node in the system. See Section 9 for details about TSP-Link.
Details	See errorqueue functions and attribute
Also see	localnode.setglobal

makegetter functions

These functions are used to set and retrieve a value for an attribute.

makegetter	
Function	Creates a function to set the value of an attribute.
Usage	`getter = makegetter(table, attribute name)` `table` Read-only table were the attribute is located. `attributename` The string name of the attribute. `getter` Function that returns the value of the given attribute.
Remarks	• 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 `getter` function will execute faster than accessing the attribute directly. • Creating a `getter` function is only useful if it is going to be called several times. Otherwise the overhead of creating the `getter` function outweighs the overhead of accessing the attribute directly.
Example	Creates a getter function called `getlevel`: `getlevel = makegetter(smua.source, "levelv")` ... v = getlevel() When `getlevel` is called, it returns the value of `smua.source.levelv`.

makesetter	
Function	Creates a function to set the value of an attribute.
Usage	`setter = makesetter(table, attributename)` `table` Read-only table where the attribute is located. attributename The string name of the attribute. setter Function that sets the value of the given attribute.
Remarks	• 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 `setter` function will execute faster than accessing the attribute directly. • Creating a `setter` function is only useful if it is going to be called several times. Otherwise the overhead of creating the `setter` function outweighs the overhead of accessing the attribute directly.

Example	Creates a `setter` function called `setlevel`:
	`setlevel = makesetter(smua.source, "levelv")`
	for v = 1, 10 do
	setlevel(v)
	end
	Using `setlevel` in the loop sets the value of `smua.source.levelv`, thereby performing a source sweep.

opc function

This function sets the OPC bit in the status register when all overlapped commands are completed.

opc	
Function	Sets the Operation Complete status bit when all overlapped commands are completed.
Usage	`opc()`
Remarks	• 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. • 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.
Details	See Appendix D.
Also see	waitcomplete

printbuffer and **printnumber** functions

These functions are used to print data and numbers.

printbuffer	
Function	Prints data from tables and reading buffer sub-tables.

Usage	There are multiple ways to use this function, the use depends on the number of tables or reading buffer subtables that are specified:
	`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)`
	`start_index` Starting index of values to print.
	`end_index` Ending index of values to print.
	`st_1, st_2, … st_n` Tables or reading buffer subtables from which to print values.

Remarks	• Correct usage when there are no outstanding overlapped commands to acquire data: • `1 <= start_index <= end_index <= n` • Where `n` refers to the index of the last entry in the tables to be printed. • If `end_index < start_index` or `n < start_index`, no data will be printed. If `start_index < 1`, 1 will be used as the first index. If `n < end_index`, `n` will be used as the last index. • 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. • When there are outstanding overlapped commands to acquire data, `n` refers to the index that the last entry in the table will have after all the measurements have completed. • If you do not specify a specific subtable in a reading buffer, then the default subtable named readings is automatically used. • At least one table or subtable must be specified. • This command generates a single response message that contains all data. The response message is stored in the output queue. • The `format.data attribute` controls the format of the response message.
Also see	format.data, printnumber

Example	This example prints all time stamps and readings in one buffer and all readings from another buffer, where n is 4:
	`format.data = format.ASCII`
	`printbuffer(1, rb1.n, rb1.timestamps, rb1, rb2)`
	Example of returned data (timestamps, rb1.readings, rb2.readings):
	`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`

printnumber	
Function	Prints numbers using the format selected for printing reading buffers.

Usage	There are multiple ways to use this function, depending on how many numbers are to be printed:
	`printnumber(v1)`
	`printnumber(v1 ,v2)`
	`printnumber(v1 ,v2, ..., vn)`
	v1, v2, ..., vn Numbers to print.

Remarks	• This function will print the given numbers using the data format specified by `format.data` and other associated attributes. • 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.
Also see	printbuffer, format.data

Example	Prints three measurements that were previously performed:
	`format.data = format.ASCII`
	`printnumber(i, v, t)`
	Example of returned data (i, v, t):
	`1.02345E-04, 8.76542E-02, 5.29372E-01`

reset function

This function is used to return all logical instruments to the default settings.

reset	
Function	Resets the logical instruments to the default settings.
Usage	`reset()`
Remarks	• 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. • Default settings are listed in Table 1-5.
Details	See "Default settings" in Section 1.

serial functions and attributes

The functions and attributes in this group are used to configure the RS-232 Interface:

serial.baud		
Attribute	Baud rate for the RS-232 port.	

Usage	`baud = serial.baud`
	`serial.baud = baud`
	baud
		-- Reads baud rate.
		-- Writes baud rate.
		Set to 300, 600, 1200, 2400, 4800, 9600, 19200 38400, 57600 or 115200.

Remarks	• A new baud rate setting takes effect when the command to change it is processed. • 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. • The baud rate is stored in non-volatile memory. The `reset` function has no effect on the baud rate.	
Details	See "RS-232 interface operation" in Section 11.	
Also see	serial.databits, serial.flowcontrol, serial.parity	
Example	Sets the baud rate to 1200: `serial.baud = 1200`	

serial.databits	
Attribute	Character width (data bits) for the RS-232 port.
Usage	`bits = serial.databits` -- Reads data width. `serial.databits = bits` -- Writes data width. bits Set to 7 or 8.
Remarks	• A new data width setting takes effect when the command to change it is processed. • 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. • The data bits value is stored in non-volatile memory. The `reset` function has no effect on data bits.
Details	See "RS-232 interface operation" in Section 11.
Also see	serial.baud, serial.flowcontrol, serial.parity
Example	Sets data width to 8: `serial.databits = 8`

serial.flowcontrol	
Attribute	Flow control for the RS-232 port.
Usage	`flow = serial.flowcontrol` -- Reads flow control. `serial.flowcontrol = flow` -- Writes flow control. Set `flow` to one of the following values: "none" or serial.FLOW_NONE Selects no flow control. "hardware" or serial.FLOW_HARDWARE Selects hardware flow control.
Remarks	• A new flow control setting takes effect when the command to change it is processed. • 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. • The flow control value is stored in non-volatile memory. The `reset` function has no effect on flow control.
Details	See "RS-232 interface operation" in Section 11.
Also see	serial.baud, serial.databits, serial.parity
Example	Sets flow control to none: `serial.flowcontrol = serial.FLOW_NONE`

serial.parity

Attribute	Parity for the RS-232 port.
Usage	`parity = serial.parity` -- Reads parity. `serial.parity = parity` -- Writes parity. Set `parity` to one of the following values: `"none"` or serial.PARITY_NONE Selects no parity. `"even"` or serial.PARITY_EVEN Selects even parity. `"odd"` or serial.PARITY_ODD Selects odd parity.
Remarks	• A new parity setting takes effect when the command to change it is processed. • 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. • The parity setting is stored in non-volatile memory. The `reset` function has no effect on parity.
Details	See "RS-232 interface operation" in Section 11.
Also see	serial.baud, serial.databits, serial.flowcontrol
Example	Sets parity to none: `serial.parity = serial.PARITY_NONE`

serial.read

Function	Reads data from the serial port.
Usage	`data = serial.read(maxchars)` maxchars Sets the maximum number of characters to read. data Returns a string consisting of all data read from the serial port.
Remarks	• This function will read available characters from the serial port. It will not wait for new characters to arrive. As long as `maxchars` 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 `maxchars`. If too many characters are received in between calls to this function, the RS-232 buffers will overflow and some characters may be lost. • 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. • 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. • 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.
Also see	serial.write

Example	Reads data from the serial port:
	`data = serial.read(200)`
	`print(data)`
	Output: `John Doe`
	The above output indicates that the string "John Doe" was read from the serial port.

serial.write

Function	Writes data to the serial port.
Usage	`serial.write(data)` data Specify the data string to write.

Remarks	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.
Also see	serial.read
Example	Writes data string "1 2 3 4" to the serial port: `serial.write("1 2 3 4")`

setup functions and attribute

The functions and attribute in this group are used to save/recall setups and to set the power-on setup.

setup.poweron	
Attribute	The saved setup to recall when the unit is turned on.
Usage	`n = setup.poweron` -- Reads the power-on setup. `setup.poweron = n` -- Writes the power-on setup. n -- Setup number to recall on power up (0 to 5).
Remarks	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.
Details	See "Remote operation setups" in Section 1.
Example	Sets unit to power on to the factory defaults: `setup.poweron = 0`

setup.recall	
Function	Recalls settings from a saved setup.
Usage	`setup.recall(n)` n Setup number to recall (0 to 5).
Remarks	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.
Details	See "Remote operation setups" in Section 1.
Example	Recalls the user-setup at location 2: `setup.recall(2)`

setup.save	
Function	Saves the present setup as a user-setup.
Usage	`setup.save(n)` n Setup number to save (1 to 5).
Remarks	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.
Details	See "Remote operation setups" in Section 1.
Example	Saves the present setup at location 5: `setup.save(5)`

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.

smuX.cal.adjustdate	X= SMU channel (a or b) (Models 2635 and 2636 only)
Attribute	Adjustment date of the last calibration adjustment

Usage	`Adjustdate = smuX.cal.adjustdate` -- Read the adjustment date. `smuX.cal.adjustdate = adjustdate` -- Write the adjustment date. Set adjustdate 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
Remarks	• This attribute stores the adjustment date associated with the active calibration set. The adjustment date can be read at any time, but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock` function. • You cannot change the adjust date with first making a change to the calibration constants. • Once you change any calibration constants, you must set the adjust date before being allowed to save the calibration data to NV memory. • This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute will reflect the date stored with that set. • Hour and minute can also be included in adjustdate as follows: `os.time({year=yr, month=mo, day=da, hour=hr, minute=mn})` Seconds can be included, but will essentially be ignored due to the precision of the internal date storage format. • The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.
Details	See Section 16 (calibration)
Also see	smuX.cal.lock, smuX.cal.unlock, smuX.cal.save, smuX.cal.restore
Example	smuX.cal.adjustdate = os.time({year = 2006, month=7, day=1})

smuX.cal.date	X = SMU channel (a or b)
Attribute	Calibration date for the active calibration set.
Usage	`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
Remarks	• This attribute stores the calibration date associated with the active calibration set. The calibration date can be read at any time but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock` function. • This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute will reflect the date stored with that set. • Hour and minute can also be included in `caldate` as follows: `os.time({year=yr, month=mo, day=da, hour=hr, minute=mn})` Seconds can be included, but will essentially be ignored due to the precision of the internal date storage format. • The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.
Details	See Section 16 "Calibration".
Also see	smuX.cal.adjustdate, smuX.cal.due, smuX.cal.restore, smuX.cal.save
Example	Sets calibration date for SMU A (July 1, 2005): `smua.cal.date = os.time({year=2005, month=7, day=1})`

smuX.cal.due	X = SMU channel (a or b)
Attribute	Calibration due date for the next calibration.

Usage	`caldue = smuX.cal.due` -- Reads calibration due date.
	`smuX.cal.due = caldue` -- Writes calibration due date.
	Set `caldue` 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`

Remarks	• This attribute stores the calibration due date associated with the active calibration set. The calibration due date can be read at any time but can only be assigned a new value when calibration has been enabled with the `smuX.cal.unlock` function. • This attribute is stored with the active calibration set. If a different calibration set is restored, this attribute will reflect the due date stored with that set. • Hour and minute can also be included in `caldate` as follows: `os.time({year=yr, month=mo, day=da, hour=hr, minute=mn})` Seconds can be included, but will be ignored due to the precision of the internal date storage format. • The date is stored as the number of seconds since UTC 12:00 am Jan 1, 1970.
Details	See Section 16 "Calibration".
Also see	smuX.cal.date, smuX.cal.restore, smuX.cal.state
Example	Sets calibration due date for SMU A (July 1, 2006): `smua.cal.due = os.time({year=2005, month=7, day=1})`

smuX.cal.lock	X = SMU channel (a or b)
Function	Disables commands that change calibration settings.
Usage	`smuX.cal.lock()`
Remarks	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 `smuX.CALSTATE_CALIBRATING`. The calibration constants must be written to non-volatile memory, or a previous calibration set must be restored prior to locking calibration.
Details	See Section 16
Also see	smuX.cal.restore, smuX.cal.save, smuX.cal.state
Example	Disable calibration functions for SMU A: `smua.cal.lock()`

smuX.cal.password	X = SMU channel (a or b)
Attribute	Password to enable calibration.
Usage	`smuX.cal.password = newpassword` `newpassword` The new password (string).
Remarks	• A new password can only be assigned when calibration has been unlocked. • The calibration password is write-only and cannot be read.
Details	See Section 16
Example	Assign a new calibration password for SMU A: `smua.cal.password = "LetMeIn"`

smuX.cal.polarity	X = SMU channel (a or b)
Attribute	Control which calibration constants are used for all subsequent measurements.
Usage	`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.
Remarks	• This attribute controls which polarity calibration constants are used to make all subsequent measurements. This attribute does not affect the `smuX.measure.calibrateY` or the `smuX.source.calibrateY` function. The polarity for those commands are dictated by the range parameter given to the command. • The measurement calibration commands require the measurements provided to have been made using the polarity being calibrated. When 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. • This attribute can only be set to positive or negative when calibration is unlocked. This attribute will automatically be set back to CAL_AUTO when calibration is locked.
Details	See Section 16 "Calibration".
Also see	smuX.measure.calibrateY, smuX.source.calibrateY
Example	Selects positive calibration constants for all subsequent measurements: `smua.cal.polarity = smua.CAL_POSITIVE`

smuX.cal.restore	X = SMU channel (a or b)
Function	Loads a stored set of calibration constants.

Usage	There are two ways to use this function:
	`smuX.cal.restore()`
	`smuX.cal.restore(calset)`
	`calset` Calibration set to be loaded.
	Set `calset` to one of the following values:
	`smuX.CALSET_NOMINAL` A set of calibration constants that are uncalibrated but set to nominal values to allow rudimentary functioning of the instrument.
	`smuX.CALSET_FACTORY` The calibration constants when the instrument left the factory.
	`smuX.CALSET_DEFAULT` The normal calibration set.
	`smuX.CALSET_PREVIOUS` The calibration set that was used before the last default set was overwritten.
	If `calset` is not specified, `smuX.CALSET_DEFAULT` will be used.

Remarks	• This function will overwrite the current set of calibration constants from non-volatile memory. • This function will be disabled until a successful call to `smuX.cal.unlock` is made.
Details	See Section 16 "Calibration".
Example	Restores factory calibration for SMU A: `smua.cal.restore(smua.CALSET_FACTORY)`

smuX.cal.save	X = SMU channel (a or b)
Function	Stores the calibration constants in non-volatile memory.
Usage	`smuX.cal.save()`
Remarks	• 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 (`smuX.CALSET_PREVIOUS`) prior to overwriting the default calibration set. • This function will be disabled until a successful call to `smuX.cal.unlock` is made. For models 2601, 2602, 2611 and 2612: 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. For models 2635 and 2636: If any of the calibration constants have been changed, this function will be disabled, unless the calibration adjust date has been set.
Details	See Section 16 "Calibration".
Also see	smuX.cal.date, smuX.cal.due, smuX.cal.restore
Example	Stores calibration constants for SMU A in non-volatile memory: `smua.cal.save()`

smuX.cal.state	X = SMU channel (a or b)
Attribute	Calibration state.
Usage	calstate = smuX.cal.state

Remarks	When reading this read-only attribute, `calstate` returns one of the following values: `0 smuX.CALSTATE_LOCKED` Calibration is locked. `1 smuX.CALSTATE_CALIBRATING` The calibration constants or dates have been changed but not yet saved to nonvolatile memory. `2 smuX.CALSTATE_UNLOCKED` Calibration is unlocked but none of the calibration constants or dates have changed since the last save/restore.
Details	See Section 16 "Calibration".
Also see	smuX.cal.due, smuX.cal.restore, smuX.cal.save

Example	Reads calibration state for SMU A:
	`calstate = smua.cal.state`
	`print(calstate)`
	Output: `0.000000e+00`
	The above output indicates that calibration is locked.

smuX.cal.unlock	X = SMU channel (a or b)
Function	Enables the commands that change calibration settings.
Usage	`smuX.cal.unlock(password)` `password` Calibration password.
Remarks	• This function enables the calibration functions to change the calibration settings. • The password when the unit is shipped from the factory is “KI0026XX”.
Details	See Section 16 "Calibration".
Also see	smuX.cal.password
Example	Unlocks calibration for SMU A: `smua.cal.unlock("KI0026XX")`

smuX.contact.calibratelo	X = SMU channel (a or b)
Function	Calibrate the low/sense low contact check measurement.
Usage	`smuX.contact.calibratehi(cp1measured, cp1reference, cp2measured, cp2reference)` `cp1measured` -- the value measured by this SMU for calibration point 1. `cp1reference` -- the reference measurement for calibration point 1 as measured externally. `cp2measured` -- the value measured by this SMU for calibration point 2. `cp2reference` -- the reference measurement for calibration point 2 as measured externally.
Remarks	• Contact check measurement calibration does not require range information. • Typically the two calibration points used will be near 0Ω for calibration point 1 and 50Ω for calibration point 2. • All four measurements (`cp1measured`, `cp1reference`, `cp2measured`, and `cp2reference`) must be made with the active calibration set. Corruption of the calibration constants may result if this is not heeded. • The new calibration constants will be activated immediately but they will not be written to non-volatile storage. Use `smuX.cal.save` to commit the new constants to nonvolatile storage. The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from non-volatile storage with the `smuX.cal.restore` function. • This function will be disabled until a successful call to `smuX.cal.unlock` is made.
Details	See Section 16 "Calibration".
Also see	smuX.contact.calibratehi

Note: 2635/36 does not have contact check

smuX.contact.calibratehi	X = SMU channel (a or b)
Function	Calibrate the high/sense high contact check measurement.
Usage	`smuX.contact.calibratehi(cp1measured, cp1reference, cp2measured, cp2reference)` `cp1measured` -- the value measured by this SMU for calibration point 1. `cp1reference` -- the reference measurement for calibration point 1 as measured externally. `cp2measured` -- the value measured by this SMU for calibration point 2. `cp2reference` -- the reference measurement for calibration point 2 as measured externally.
Remarks	• Contact check measurement calibration does not require range information. • Typically the two calibration points used will be near 0Ω for calibration point 1 and 50Ω for calibration point 2. • All four measurements (`cp1measured`, `cp1reference`, `cp2measured`, and `cp2reference`) must be made with the active calibration set. Corruption of the calibration constants may result if this is not heeded. • The new calibration constants will be activated immediately but they will not be written to non-volatile storage. Use `smuX.cal.save` to commit the new constants to nonvolatile storage. The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from non-volatile storage with the `smuX.cal.restore` function. • This function will be disabled until a successful call to `smuX.cal.unlock` is made.
Details	See Section 16 "Calibration".
Also see	smuX.contact.calibratelo

Note: 2635/36 does not have contact check

smuX.contact.check	X = SMU channel (a or b)
Function	Determine if contact resistance is lower than threshold.
Usage	`flag = smuX.contact.check()` -- check contact resistance against threshold
Remarks	Attempting to perform a contact check measurement when any of the following conditions exists will generate an error: • Output is off in High-Z mode. • Current limit set to less than 1mA.
Details	See Section 3 for connections.
Also see	smuX.contact.threshold, smuX.contact.speed

Example	Takes action if contact check on SMU A fails:
	`if (not smua.contact.check()) then`
	`  -- take action`
	`end`

Note: 2635/36 does not have contact check

smuX.contact.r	X = SMU channel (a or b)
Function	Measure contact resistance.
Usage	`rhi, rlo = smuX.contact.r()` `rhi` – the measured contact resistance on the high/sense high side rlo – the measured contact resistance on the low/sense low side.

Remarks	Attempting to perform a contact check measurement when any of the following conditions exists will generate an error: • Output is off in High-Z mode. • Current limit set to less than 1mA.
Details	See Section 3 for connections.
Also see	smuX.contact.speed

Note: 2635/36 does not have contact check

smuX.contact.speed

X = SMU channel (a or b)

Function	The speed setting for contact check measurements.

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

Remarks	This setting controls the aperture of measurements made for contact check. It does not affect the `smuX.measure.nplc` aperture setting. The speed setting can have a dramatic effect on the accuracy of the measurement, as reflected in the specifications.
Details	See Section 3 for connections.
Example	Set contact check measurements on SMU A for higher accuracy: `smua.contact.speed = smua.CONTACT_SLOW`

Note: 2635/36 does not have contact check

smuX.contact.threshold

X = SMU channel (a or b)

Function	Resistance threshold for the smuX.contact.check function.
Usage	smuX.contact.threshold = rvalue -- writes the threshold rvalue = smuX.contact.threshold -- reads the threshold rvalue -- Set to the resistance, in ohms, above which contact check should fail
Remarks	The default threshold is 50Ω. The threshold should be set to less than 1kΩ.
Details	See Section 3 for connections.
Also see	smuX.contact.check
Example	Set the contact check threshold for SMU A to 5 Ω: `smua.contact.threshold = 5`

Note: 2635/36 does not have contact check

smuX.makebuffer

X = SMU channel (a or b)

Function	Creates a RAM buffer.
Usage	`mybuffer = smuX.makebuffer(buffersize)` `buffersize` Number of readings that can be stored.
Remarks	• RAM reading buffers can be allocated dynamically. These are created and allocated with the `smuX.makebuffer(buffer)` function, where `buffersize` is the number of readings the buffer can store. • Dynamically allocated reading buffers can be used interchangeably with the `smuX.nvbufferY` buffers. • A RAM buffer can be deleted using `nil`. The following command deletes `mybuffer`: • `mybuffer = nil`
Details	See "Reading buffers" in Section 7.

Also see	smuX.nvbufferY
Example	Creates a 200 reading RAM buffer named “mybuffer2” for SMUA: `mybuffer2 = smua.makebuffer(200)`

smuX.measure.analogfilter	X = SMU channel (a or b) (Models 2635 and 2636 only)
Attribute	Controls the use of an analog filter when measuring on the lowest current ranges.
Usage	`option = smuX.measure.analogfilter` -- Reads the filter setting. `smuX.measure.analogfilter = option` -- Writes the filter setting. where option is: 0 filter off 1 filter on
Remarks	• This attribute engages an approximately 1hz analog filter across the current range elements. • The analog filter is only active when using the 1nA and 100pA measurement ranges.
Example	`smua.measure.analogfilter = 0` - -Turn off analog filter.

smuX.measure.autorangeY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Attribute	Measure auto range setting.
Usage	`autorange = smuX.measure.autorangeY` -- Reads measure auto range. `smuX.measure.autorangeY = autorange` -- Writes measure auto range. Set `autorange` to one of the following values: `0` or `smuX.AUTORANGE_OFF` Disables measure auto range. `1` or `smuX.AUTORANGE_ON` Enables measure auto range.
Remarks	• This attribute indicates the measurement auto range state. Its value will be `smuX.AUTORANGE_OFF` when the SMU measure circuit is on a fixed range and `smuX.AUTORANGE_ON` when it is in auto range mode. • Setting this attribute to `smuX.AUTORANGE_OFF` puts the SMU on a fixed range. The fixed range used will be the range the SMU measure circuit was currently using. • Setting this attribute to `smuX.AUTORANGE_ON` puts the SMU measure circuit into auto range mode. It will remain on its present measure range until the next measurement is requested.
Details	See “Range” in Section 6.
Also see	smuX.measure.rangeY
Example	Enables voltage measure autoranging for SMU A: `smua.measure.autorangev = smua.AUTORANGE_ON`

smuX.measure.autozero	X = SMU channel (a or b)
Attribute	Behavior of the SMU’s A/D internal reference measurements (autozero).
Usage	azmode = smuX.measure.autozero -- Reads autozero. smuX.measure.autozero = azmode -- Writes autozero. Set `azmode` to be one of the following values: `0` or `smuX.AUTOZERO_OFF` Autozero disabled. `1` or `smuX.AUTOZERO_ONCE` Performs autozero once, then disables autozero. `2` or `smuX.AUTOZERO_AUTO` Automatic checking of reference and zero measurements. An autozero is performed when needed.

Remarks	• The Series 2600 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. • By default, the instrument automatically checks these reference measurements whenever a signal measurement is made. If the reference measurements have expired when a signal measurement is made, the instrument will automatically take two more A/D conversions, one for the reference and one for the zero, before returning the result. Thus, occasionally, a measurement takes longer than normal. This 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 `smuX.measure.autozero` 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. To minimize the drift, a reference and zero measurement should be made just prior to the critical test sequence. `The smuX.AUTOZERO_ONCE` setting can be used to force a refresh of the reference and zero measurements used for the current aperture setting. • Autozero reference measurements for the last 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.
Example	Perform autozero once for SMU A: `smua.measure.autozero = smua.AUTOZERO_ONCE`

smuX.measure.calibrateY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Function	Generates and activates new measurement calibration constants.
Usage	`smuX.measure.calibrateY(range, cp1measured, cp1reference, cp2measured, cp2reference)` `range` The measurement range to calibrate. `cp1measured` The value measured by this SMU for calibration point 1. `cp1reference` The reference measurement for calibration point 1 as measured externally. `cp2measured` The value measured by this SMU for calibration point 2. `cp2reference` The reference measurement for calibration measured externally.

Remarks	• This function generates and activates new calibration constants for the given range. The positive and negative polarities of the instrument must be calibrated separately. Use a positive value for `range` to calibrate the positive polarity and a negative value for `range` to calibrate the negative polarity. • Typically the two calibration points used will be near zero for calibration point 1 and 90% of full scale for calibration point 2. • All four measurements (`cp1measured`, `cp1reference`, `cp2measured`, and `cp2reference`) **must** be made with the active calibration set. Corruption of the calibration constants may result if this is not heeded. • The new calibration constants will be activated immediately but they will not be written to non-volatile storage. Use `smuX.cal.save` to commit the new constants to nonvolatile storage. The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from non-volatile storage with the `smuX.cal.restore` function. • This function will be disabled until a successful call to `smuX.cal.unlock` is made.
Details	See Section 16 "Calibration".
Also see	smuX.cal.restore, smuX.cal.save, smuX.makebuffer, smuX.source.calibrateY

smuX.measure.count

X = SMU channel (a or b)

Attribute	Number of measurements performed when a measurement is requested.
Usage	`count = smuX.measure.count` -- Reads measure count. `smuX.measure.count = count` -- Writes measure count. count Number of measurements.
Remarks	• This attribute controls the number of measurements taken any time a measurement is requested. When using a reading buffer with a measure command, the `count` also controls the number of readings to be stored. • The `reset` function sets the measure count to 1.
Details	See "Triggering" in Section 10.
Also see	smuX.measure.overlappedY, smuX.measure.Y
Example	Sets measure count for SMU A: smua.measure.count = 10

smuX.measure.delay

X= SMU channel (a or b)
(Models 2635 and 2636 only)

Attribute	Controls the measurement delay.
Usage	`mdelay = smuX.measure.delay` -- Read the measure delay. `smuX.measure.delay = mdelay` -- Write the measure delay. Set `mdelay` to one of the following values: `0` or `smuX.DELAY_OFF` No delay. `-1` or `smux.DELAY_AUTO` Automatic delay value. `user_value` Set user delay value.
Remarks	• This attribute allows for additional settling time before taking a measurement. • The default is `smuX.DELAY_AUTO`. • The `smuX.DELAY_AUTO` setting causes a current range-dependent delay to be inserted when a current measurement function is called. • If `smuX.measure.count` is greater than 1, the measurement delay is only inserted before the first measurement. • `mdelay` can be set to a specific user-defined value that sets the delay that is used regardless of range. • Use `smuX.source.delay` when measuring your source (source read back) and `smuX.measure.delay` when stepping voltage and measuring current.
Also see	smuX.measure.delayfactor, smuX.source.delay
Example	`smua.measure.delay = 0.010` Sets the measurement delay to 10mS.

smuX.measure.delayfactor	X= SMU channel (a or b) (Models 2635 and 2636 only)
Attribute	This attribute is a multiplier to smuX.DELAY_AUTO measurement delays.
Usage	`delayfactor = smuX.measure.delayfactor` --Read the delay factor. `smuX.measure.delayfactor = delayfactor` --Set the delay factor.
Remarks	• The delay factor is only applied when `smuX.measure.delay = smuX.DELAY_AUTO`. • The default value is 1.0. • This attribute can be set to a value less than 1 (e.g. 0.5) to decrease the automatic delay. • This attribute can be set to a value greater than 1 (e.g. 1.5 or 2.0) to increase the automatic delay. • Setting this attribute to zero disables delays, even when `smuX.measure.delay = smuX.DELAY_AUTO`.
Also see	smuX.measure.delay
Example	`smuX.measure.delayfactor = 2.0` Increase the automatic delay by 2 times.

smuX.measure.filter.count	X = SMU channel (a or b)
Attribute	Number of measured readings to yield one filtered measurement.
Usage	count = smuX.measure.filter.count -- Reads filter count. smuX.measure.filter.count = count -- Writes filter count. count Set filter count from 1 to 100.
Remarks	• This attribute is the number of measurements that will be performed to yield one filtered measurement. • The `reset` function sets the filter count to 1.
Details	See "Filters" in Section 6.
Also see	smuX.measure.filter.enable, smuX.measure.filter.type
Example	Sets filter count for SMU A: smua.measure.filter.count = 10

smuX.measure.filter.enable	X = SMU channel (a or b)
Attribute	Enables/disables filtered measurements.
Usage	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.
Remarks	• This attribute enables or disables the filter. • The `reset` function disables the filter. • The median filter uses a first-in, first-out stack. When the stack (filter count) becomes full, the "middle-most" reading is returned. For each subsequent conversion placed into the stack, the oldest reading is discarded. The stack is then re-sorted, yielding a new reading. If the filter count is an even number, the reading returned is the average of the two middle readings.
Details	See "Filters" in Section 6.
Also see	smuX.measure.filter.count, smuX.measure.filter.type • The median filter uses a first-in, first-out stack. When the stack (filter count) becomes full, the "middle-most" reading is returned. For each subsequent conversion placed into the stack, the oldest reading is discarded. The stack is then re-sorted, yielding a new reading. If the filter count is an even number, the reading returned is the average of the two middle readings.

Example	Enable the filter for SMU A: `smua.measure.filter.enable = smua.FILTER_ON`

smuX.measure.filter.type	X = SMU channel (a or b)
Attribute	Type of filter for measurements.
Usage	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. 2 or smuX.FILTER_MEDIAN Selects the median filter (Models 2635 and 2636 only).
Remarks	• 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. • The `reset` function selects the repeat filter.
Details	See "Filters" in Section 6.
Also see	smuX.measure.filter.count, smuX.measure.filter.enable
Example	Selects the moving average filter for SMU A: `smua.measure.filter.type = smua.FILTER_MOVING_AVG`

smuX.measure.interval	X = SMU channel (a or b)
Attribute	Interval between multiple measurements.
Usage	`interval` = smuX.measure.`interval` -- Reads measure interval. smuX.measure.`interval` = `interval` -- Writes measure interval. interval Set interval (in seconds) from 0 to 1.
Remarks	• This attribute sets the time interval between groups of measurements when `smua.measure.count` is set to a value greater than 1. The SMU will do its best to start the measurement of each group when scheduled. • 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. • If the SMU cannot keep up with the interval setting, measurements will be made as fast as possible. • The `reset` function sets the measure interval to 0.
Details	See "Triggering" in Section 10.
Also see	smuX.measure.overlappedY, smuX.measure.Y
Example	Sets measure interval for SMU A: `smua.measure.interval = 0.5`

smuX.measure.lowrangeY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Attribute	Lowest measure range that will be used during autoranging.
Usage	`rangeval` = smuX.measure.`lowrangeY` -- Reads low range. smuX.measure.`lowrangeY` = `rangeval` -- Writes low range. `rangeval` Set to the lowest voltage or current measure range.
Remarks	• 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. • 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.
Details	See "Range" in Section 6.
Also see	smuX.measure.autorangeY
Example	Sets volts lowrange for Model 2601/2602 SMU A to 1V: smua.measure.`lowrangev` = `1`

smuX.measure.nplc	X = SMU channel (a or b)
Attribute	Integration aperture for measurements.
Usage	`nplc` = smuX.measure.`nplc` -- Reads nplc. smuX.measure.`nplc` = `nplc` -- Writes nplc. `nplc` Set from 0.001 to 25.
Remarks	• 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). • The reset function sets the aperture to 1.0.
Details	See "Speed" in Section 6.
Example	Sets integration time for SMU A (0.5/60 seconds): smua.measure.`nplc` = `0.5`

smuX.measure.overlappedY **smuX.measure.overlappediv**	X = SMU channel (a or b) Y = SMU measure function (v, i, r or p) where: v = voltage, i = current, r = resistance, p = power
Function	Starts an asynchronous (background) measurement.

Usage	There are two ways to use this function:
	`smuX.measure.overlappedY(rbuffer)`
	`smuX.measure.overlappediv(ibuffer, vbuffer)`
	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.

Remarks	• 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. • The `smuX.measure.overlappediv` function stores both current and voltage readings in respective buffers (current and then voltage are stored in separate buffers). • This function is an overlapped command. Script execution will continue while the measurement(s) is made in the background. Attempts to access result values that have not yet been generated will cause the script to block and wait for the data to become available. The `waitcomplete` function can also be used to wait for the measurement(s) to complete before continuing. • If a given reading buffer contains any data, it will be cleared prior to taking any measurements, unless the reading buffer has been configured to append data.
Details	See "Reading buffers" in Section 7.
Also see	smuX.nvbufferY, smuX.nvbufferY, waitcomplete
Example	Starts background voltage measurements for SMU A: `smua.measure.overlappedv(smua.nvbuffer1)`

smuX.measure.rangeY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Attribute	Fixed measure range for voltage or current.
Usage	`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.
Remarks	• Reading this attribute returns the positive full-scale value of the measure range the SMU is currently using. • 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`. • 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 Model 2601/2602 6V range or measuring 5.0V on the Model 2611/2612 2V range will cause an over-range. The value 9.91000E+37 is returned when this occurs. • If the source function is the same as the measurement function (for example, sourcing voltage and measuring voltage), the measurement range is locked to be the same as the source range. However, the setting for the voltage measure range is retained and used when the source function is changed to current, and the present voltage measurement range will be used. • Model 2601/2602 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. • 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. • 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. • 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.
Details	See “Range” in Section 6.
Also see	smuX.measure.autorangeY
Example	Selects 1V measure range for Model 2601/2602 SMU A: `smua.measure.rangev = 0.5`

smuX.measure.rel.enableY	X = SMU channel (a or b) Y = SMU measure function (v, i, r or p) where: v = voltage, i = current, r = resistance, p = power
Attribute	Relative measurement control (on/off).
Usage	`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.
Remarks	When relative measurements are enabled, all subsequent measured readings will be offset by the specified relative offset value (see `smuX.measure.rel.levelY`). Specifically, each returned measured relative reading will be the result of the following calculation: Relative reading = Actual measured reading – Relative offset value
Details	See "Rel" in Section 6.
Also see	smuX.measure.rel.levelY
Example	Enables relative voltage measurements for SMU A: `smua.measure.rel.enablev = smua.REL_ON`

smuX.measure.rel.levelY	X = SMU channel (a or b) Y = SMU measure function (v, i, r or p)where: v = voltage, i = current, r = resistance, p = power
Attribute	Offset value for relative measurements.
Usage	`relval = smuX.measure.rel.levelY` -- Reads relative offset level. `smuX.measure.rel.levelY = relval` -- Writes relative offset level. `relval` -- Relative offset value.
Remarks	When relative measurements are enabled (see smuX.measure.rel.enableY), 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: • Relative reading = Actual measured reading – Relative offset value
Details	See "Rel" in Section 6.
Also see	smuX.measure.rel.enableY
Example	Performs a voltage measurement and uses it as the relative offset value: `smua.measure.rel.levelv = smua.measure.v()`

smuX.measure.Y **smuX.measure.iv**	X = SMU channel (a or b) Y = SMU measure function (v, i, r or p) where: v = voltage, i = current, r = resistance, p = power
Function	Performs one or more measurements.

Usage	There are three ways to use this function:
	`reading` = smuX.measure.`Y()`
	`reading` = smuX.measure.`Y(rbuffer)`
	reading = smuX.measure.iv(ibuffer, vbuffer)
	reading — Returns the last reading of the measurement process.
	rbuffer — A reading buffer object where all 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.

Remarks	• This function returns only the last actual measurement as `reading`. 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 `rbuffer` if one is provided, but only the last measurement will be returned as `reading`. • The `smuX.measure.iv` function stores both current and voltage readings in respective buffers (current and then voltage are stored in separate buffers). • The smuX.measure.count attribute determines how many measurements are performed. When using a buffer, it also determines the number of readings to store in the buffer.
Details	See "Reading buffers" in Section 7.
Also see	smuX.nvbufferY, smuX.nvbufferY

Example	Performs ten voltage measurements using SMU A and stores them in a buffer:
	`smua.measure.count = 10`
	`smua.measure.v(smua.nvbuffer1)`

smuX.measureYandstep **smuX.measureivandstep**	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
Function	Performs one or two measurements and then steps the source.

Usage	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.

Remarks	• 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). • The specified source value should be appropriate for the selected source function. For example, if the source voltage function is selected, then `sourcevalue` is expected to be a new voltage level. • Both source and measure auto range must be disabled before using this function. • 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.
Also see	smuX.measure.Y

Example	This Model 2601/2602 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.
	`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()`

smuX.nvbufferY	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Attribute	Non-volatile reading buffers.
Usage	`smuX.nvbufferY`
Remarks	• There are two reading buffers: `smuX.nvbuffer1` and `smuX.nvbuffer2`. • 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. • The non-volatile reading buffers will retain their data between power cycles.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.makebuffer, smuX.measure.overlappedY, smuX.measure.Y
Example	Store current readings from SMU A into buffer 1: smua.measure.overlappedi(smua.nvbuffer1)

smuX.nvbufferY.appendmode	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Attribute	Append mode for the reading buffer.
Usage	state = smuX.nvbufferY.appendmode -- Reads append mode. smuX.nvbufferY.appendmode = state -- Writes append mode. Set `state` to one of the following values: `0` Append mode off — New measure data overwrites the previous buffer content. `1` Append mode on — Appends new measure data to the present buffer content.
Remarks	• Assigning to this attribute enables or disables the buffer append mode. • With append mode on, the first new measurement will be stored at `rb[n+1]`, where `n` is the number of readings stored in the buffer.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
Example	Append new readings for SMU A to contents of buffer 1: `smua.nvbuffer1.appendmode = 1`

smuX.nvbufferY.basetimestamp	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Attribute	Timestamp of when the first reading was stored from time of power-up.
Usage	`basetime = smuX.nvbufferY.basetimestamp`

Remarks	• Reading this attribute returns the timestamp (in seconds) for the first reading (`rb[1]`) stored in a buffer. The timestamp is based on the number of seconds from power-up that the measurement was performed and stored. • This is a read-only attribute.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY

Example	Read the timestamp for the first reading stored in buffer 1 of SMU A:
	`basetime = smua.nvbuffer1.basetimestamp`
	`print(basetime)`
	Output: `2.369900+03`
	The above output indicates that the timestamp is 2369.9 seconds.

smuX.nvbufferY.capacity	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Attribute	Capacity of the buffer.
Usage	`capacity = smuX.nvbufferY.capacity`
Remarks	• Reading this attribute returns the number of readings that can be stored in the buffer. • 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. • This is a read-only attribute.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
Example	Read the capacity of SMU A buffer 1: capacity = smua.nvbuffer1.capacity print(capacity) Output: `1.123410+05`

smuX.nvbufferY.clear	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Function	Clears the buffer.
Usage	`smuX.nvbufferY.clear()`
Remarks	This function clears all readings from the indicated buffer.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
Example	Clears SMU A buffer 1: smua.nvbuffer1.clear()

smuX.nvbufferY.collectsourcevalues	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Attribute	Source value collection for the buffer.
Usage	state = smuX.nvbufferY.collectsourcevalues -- Reads collection state. smuX.nvbufferY.collectsourcevalues = state -- Writes collection state. Set `state` to one of the following values: `0` Source value collection disabled (off). `1` Source value collection enabled (on).
Remarks	• Assigning a `state` value to this attribute enables or disables the storage of source values. Reading this attribute returns the `state` of source value collection. • When on, source values will be stored with readings in the buffer. This requires four extra bytes of storage per reading. • This value, off or on, can only be changed when the buffer is empty. The buffer can be emptied using the smuX.nvbufferY.clear function.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY

Example	Include source values with readings for SMU A buffer 1: `smua.nvbuffer1.collectsourcevalues = 1`

smuX.nvbufferY.collecttimestamps	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Attribute	Timestamp collection for the buffer.
Usage	state = smuX.nvbufferY.collecttimestamps-- Reads collection state. smuX.nvbufferY.collecttimestamps = state-- Writes collection state. Set `state` to one of the following values: `0` Timestamp collection disabled (off). `1` Timestamp collection enabled (on).
Remarks	• Assigning a `state` value to this attribute enables or disables the storage of timestamps. Reading this attribute returns the `state` of timestamp collection. • When on, timestamps will be stored with readings in the buffer. This requires four extra bytes of storage per reading. The first reading is time stamped at zero seconds. Subsequent readings are time stamped relative to the time storage was started. • This value, off or on, can only be changed when the buffer is empty. The buffer can be emptied using the smuX.nvbufferY.clear function.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
Example	Include timestamps with readings for SMU A buffer 1: smua.nvbuffer1.collecttimestamps = 1

smuX.nvbufferY.n	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Attribute	Number of readings in the buffer.
Usage	`bufferreadings = smuX.nvbufferY.n`
Remarks	• Reading this attribute returns the number of readings that are stored in the buffer. • This is a read-only attribute.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY

Example	Read the number of readings stored in SMU A buffer 1:
	`bufferreadings = smua.nvbuffer1.n`
	`print(bufferreadings)`
	Output: `1.250000+02`
	The above output indicates that there are 125 readings stored in the buffer.

smuX.nvbufferY.timestampresolution	X = SMU channel (a or b) Y = NV buffer (1 or 2)
Attribute	Timestamp resolution.
Usage	tsres = smuX.nvbufferY.timestampresolution -- Reads collection state. smuX.nvbufferY.timestampresolution = tsres -- Writes collection state. `tsres` Timestamp resolution in seconds.
Remarks	• Assigning to this attribute sets the resolution for the timestamps. Reading this attribute returns the timestamp resolution value. • The minimum 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. • When setting this value it will be rounded to an even power of 2µs.
Details	See "Reading buffers" in this section and in Section 7.
Also see	smuX.measure.overlappedY, smuX.measure.Y, smuX.nvbufferY
Example	Set the timestamp resolution for SMU A buffer 1 to 10µs: `smua.nvbuffer1.timestampresolution = 0.00001`

smuX.reset	X = SMU channel (a or b)
Function	Turns off the output and resets the SMU to the default settings.
Usage	smuX.reset()
Remarks	Returns the SMU to the default settings listed in Table 1-5.
Details	See "Default settings" in Section 1.
Also see	reset

smuX.sense	X = SMU channel (a or b)
Attribute	Remote/local sense mode.
Usage	`sense` = smuX.`sense` -- Reads sense mode. smuX.`sense = sense` -- Writes sense mode. Set `sense` to one of the following values: `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.
Remarks	• Source-measure operations are performed using either 2-wire local sense connections or 4-wire remote sense connections. Writing to this attribute selects the sense mode. • The `smuX.SENSE_CALA` mode is only used for calibration and may only be selected when calibration is enabled. • The sense mode can be changed between local and remote while the output is on. • The calibration sense mode cannot be selected while the output is on. • The `reset` function selects the local sense mode.
Details	See "Sensing methods" in Section 3.
Example	Selects remote sensing for SMU A: `smua.sense = smua.SENSE_REMOTE`

smuX.source.autorangeY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Attribute	Source auto range control (on/off).
Usage	`sautorange = smuX.source.autorangeY` -- Reads source auto range. `smuX.source.autorangeY = sautorange` -- Writes source auto range. Set `autorange` to one of the following values: `0` or `smuX.AUTORANGE_OFF` Disables source auto range. `1` or `smuX.AUTORANGE_ON` Enables source auto range.
Remarks	• This attribute indicates the source auto range state. Its value will be `smuX.AUTORANGE_OFF` when the SMU source circuit is on a fixed range and `smuX.AUTORANGE_ON` when it is in auto range mode. • Setting this attribute to `smuX.AUTORANGE_OFF` puts the SMU on a fixed source range. The fixed range used will be the range the SMU source circuit was currently using. • Setting this attribute to `smuX.AUTORANGE_ON` 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. • Auto range will disable if the source level is edited from the front panel.
Details	See "Range" in Section 6.
Also see	smuX.measure.autorangeY, smuX.source.rangeY
Example	Enables volts source auto range for SMU A: `smua.source.autorangev = smua.AUTORANGE_ON`

smuX.source.calibrateY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Function	Generates and activates new source calibration constants.
Usage	`smuX.source.calibrateY(range, cp1expected, cp1reference, cp2expected, cp2reference)` `range` The measurement range to calibrate. `cp1expected` The source value programmed for calibration point 1. `cp1reference` The reference measurement for calibration point 1 as measured externally. `cp2expected` The source value programmed for calibration point 2. `cp2reference` The reference measurement for calibration point 2 as measured externally.
Remarks	• This function generates and activates new calibration constants for the given range. The positive and negative polarities of the source must be calibrated separately. Use a positive value for `range` to calibrate the positive polarity and a negative value for `range` to calibrate the negative polarity. • 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. • Do not use 0.0 for a negative calibration point as 0.0 is considered a positive number. • The two reference measurements must be made with the source using the active calibration set. For example, source a value, measure it, and do not change the active calibration set before issuing this command. • The new calibration constants will be activated immediately but they will not be written to non-volatile storage. Use `smuX.cal.save` to commit the new constants to nonvolatile storage. The active calibration constants will stay in effect until the instrument is power cycled or a calibration set is loaded from non-volatile storage with the `smuX.cal.restore` function. • This function will be disabled until a successful call to `smuX.cal.unlock` is made.
Details	See Section 16 "Calibration".
Also see	smuX.cal.restore, smuX.cal.save, smuX.makebuffer, smuX.measure.calibrateY

smuX.source.compliance	X = SMU channel (a or b)
Attribute	Source compliance state.
Usage	`compliance = smuX.source.compliance`
Remarks	• Use this attribute to read the state of source compliance. true indicates that the limit function is in control of the source (source in compliance). false indicates that the source function is in control of the output (source not in compliance). • This is a read-only attribute. Writing to this attribute will generate an error. • Reading this attribute also updates the status model and the front panel with generated compliance information.
Details	See Section 4 “Basic operation” and Appendix D “Status model”.
Also see	smuX.source.limitY
Example	Reads the source compliance state for SMU A: compliance = smua.source.compliance print(compliance) Output: true 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).

smuX.source.delay	X = SMU channel (a or b)
Attribute	Source delay.
Usage	`smuX.source.delay = Y` -- Writes source delay. `Y = smuX.source.delay` --Read source delay. Set `Y` to one of the following values: `0` or `smuX.DELAY_OFF` No delay. `-1` or `smuX.DELAY_AUTO` Auto delay. `User_value` Set user delay value.
Remarks	• This attribute allows for additional source settling time after an output step. • The default is 0, no delays. • Setting this attribute to `DELAY_AUTO` will cause a range dependent delay to be inserted when ever the source is changed. • Y can be set to a specific user defined value that will set the delay that is used regardless of range.
Example	Selects the delay to auto SMU A: `smua.source.delay = smua.DELAY_AUTO`

smuX.source.func	X = SMU channel (a or b)
Attribute	Source function.
Usage	`iv = smuX.source.func` -- Reads source function. `smuX.source.func = iv` -- Writes source function. Set `iv` to one of the following values: `0` or `smuX.OUTPUT_DCAMPS` Selects current source function. `1` or `smuX.OUTPUT_DCVOLTS` Selects voltage source function.
Remarks	• 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. • The `reset` function selects the voltage function.
Details	See Section 4 "basic operation".
Also see	smuX.source.output, smuX.source.levelY
Example	Selects the source amps function for SMU A: `smua.source.func = smua.OUTPUT_DCAMPS`

smuX.source.levelY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Attribute	Source levels.
Usage	`sourceval = smuX.source.levelY`-- Reads source value. `smuX.source.levelY = sourceval`-- Writes source value. `sourceval` Set 2601/2602 voltage from 0 to ±40 (volts). Set 2601/2602 current from 0 to ±3 (amps). Set 2611/2612/2635/2636 voltage from 0 to ±200 (volts). Set 2611/2612/2635/2636 current from 0 to ±1.5 (amps).
Remarks	• This attribute configures the source level of the voltage or current source. • If the source is configured as a voltage source and the output is on, the new `smuX.source.levelv` setting will be sourced immediately. If the output is off or 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. • If the source is configured as a current source and the output is on, the new `smuX.source.leveli` setting will be sourced immediately. If the output is off or 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. • The sign of `sourceval` dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal. • The `reset` function sets the source levels to 0V and 0A.
Details	See Section 4 "basic operation"
Also see	smuX.source.func, smuX.source.output
Example	Sets V-source to 1V for SMU A: `smua.source.levelv = 1`

smuX.source.limitY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Attribute	Compliance limits.
Usage	`limit = smuX.source.limitY` -- Reads compliance limit. `smuX.source.limitY = limit` -- Writes compliance limit. `limit` 2601/2602 voltage compliance from 0 to ±40 (volts). 2601/2602 current compliance from 0 to ±3 (amps). 2611/2612/2635/2636 voltage compliance from 0 to ±200 (volts). 2611/2612/2635/2636 current compliance from 0 to ±1.5 (amps).
Remarks	• Use the `smuX.source.limiti` attribute to limit the current output of the voltage source. Use `smuX.source.limitv` to limit the voltage output of the current source. The SMU will always choose (auto range) the source range for the limit setting. • This attribute should be set in the test sequence before the turning the source on. • Reading this attribute indicates the presently set compliance value. Use `smuX.source.compliance` to read the state of source compliance.
Details	See Section 4 "basic operation".
Also see	smuX.source.compliance, smuX.source.func, smuX.source.output
Example	Sets V-compliance to 30V for SMU A: `smua.source.limitv = 30`

smuX.source.lowrangeY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Attribute	Lowest source range that will be used during autoranging.
Usage	`rangeval` = smuX.source.`lowrangeY`-- Reads low range. smuX.source.`lowrangeY` = `rangeval`-- Writes low range. `rangeval` Set to the lowest voltage or current to be sourced.
Remarks	• This attribute is used with source autoranging to put a lower bound on the range used. Lower ranges generally require greater settling times. By setting a low range value, sourcing small values might be able to be made with less settling time. • If the instrument is set to auto range and it is on a range lower than the one specified by `rangeval`, the source range will be changed to the range specified by `rangeval`.
Details	See "Range" in Section 6.
Also see	smuX.source.autorangeY, smuX.source.rangeY
Example	Sets volts lowrange for Model 2601/2602 SMU A to 1V. This prevents the source from using the 100mV range when sourcing voltage: `smua.source.lowrangev = 1`

smuX.source.offlimiti	X = SMU channel (a or b)
Attribute	The current limit used when the SMU is in output off normal mode.
Usage	`ivalue = smuX.source.offlimiti` -- read the limits `smuX.source.offlimiti = ivalue` -- write the limit `ivalue` = the current limit to use
Remarks	Setting this limit to lower than 1mA will not allow the contact check function to operate when the output is off in normal mode.
Details	See "Output-off states" in Section 3.
Also see	smuX.source.offmode
Example	Change the off normal limit to 10mA for SMU A: smua.source.offlimiti = 10e-3

smuX.source.offmode	X = SMU channel (a or b)
Attribute	Source output-off mode.
Usage	`offmode` = smuX.source.`offmode` -- Reads output-off mode. smuX.source.`offmode` = `offmode` -- Writes output-off mode. Set `offmode` to one of the following values: `0` or smuX.OUTPUT_NORMAL — Outputs 0V when the output is turned off. `1` or smuX.OUTPUT_ZERO — Zero the output (in either volts or current) when off. `2` or smuX.OUTPUT_HIGH_Z — Opens the output relay when the output is turned off.

Remarks	• Reading this attribute gives the output-off mode of the source. Setting this attribute configures the SMU output-off mode. • The default `offmode` is `smuX.OUTPUT_NORMAL`. In this mode, the SMU will source 0 volts. For the 2601/2602, 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. For the 2611/2612/2635/2636, the compliance to the value specified by `smuX.source.offlimiti` (default 1mA). • When offmode is set to `smuX.OUTPUT_HIGH_Z`, the SMU will open the output relay when the output is turned off. • When the offmode is set to `smuX.OUTPUT_ZERO`, the SMU will source 0 volts just as `OUTPUT_NORMAL` 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.
Details	See "Output-off states" in Section 3.
Also see	smuX.source.output
Example	Sets output-off mode for SMU A: `smua.source.offmode = smua.OUTPUT_HIGH_Z`

smuX.source.output X = SMU channel (a or b)

Attribute	Source output control (on/off).
Usage	`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.
Remarks	Reading this attribute gives the output state of the source. Setting this attribute will turn the output of the source on or off. The default for the source is off. When the output is turned on, the SMU will source either voltage or current as dictated by the `smuX.source.func` setting.
Details	See Section 4 "basic operation".
Also see	smuX.source.func, smuX.source.offmode
Example	Turns SMU A source output on: smua.source.`output` = smua.OUTPUT_ON

smuX.source.outputenableaction X = SMU channel (a or b)

Attribute	Output enable or interlock action for the source.
Usage	`action` = smuX.source.`output`enableaction-- Reads enable action. smuX.source.`output`enableaction = action-- Writes enable action. Set `action` to one of the following values: `0` or smuX.OE_NONE No action. `1` or smuX.OE_OUTPUT_OFF Turns the source output off.

Remarks	• This attribute controls the SMU action taken when the output enable or interlock line is asserted/de-asserted. The default setting is `smuX.OE_NONE`. • This attribute is always set to `OE_OUTPUT_OFF` for the Model 2611/2612 and is read-only. • When set to `smuX.OE_NONE`, the SMU will take no action when the output enable or interlock line goes low (de-asserted). • When set to `smuX.OE_OUTPUT_OFF` and the output enable or interlock line is de-asserted, the SMU will turn its output off as if the `smuX.source.output = smuX.OUTPUT_OFF` command had been received. • The SMU will not automatically turn its output on when the output enable or interlock line returns to the high state. • If the output enable or interlock line is not asserted when this attribute is set to `smuX.OE_OUTPUT_OFF` and the output is on, the output will turn off immediately. • Detection of the output enable or interlock line going low will not abort any running scripts. This may cause execution errors.
Details	See "Output Enable (Models 2601/2602)" in Section 10.
Also see	smuX.source.offmode, smuX.source.output
Example	Reconfigures the SMU to turn the output off if the output enable or interlock line goes low (de-asserted): `smua.source.outputenableaction = smua.OE_OUTPUT_OFF`

smuX.source.rangeY	X = SMU channel (a or b) Y = SMU measure function (v or i) where v = voltage, i = current
Attribute	Source range.
Usage	`rangeval` = smuX.source.`rangeY` -- Reads source range. smuX.source.`rangeY` = `rangeval` -- Writes source range. `rangeval` The expected voltage or current to be sourced.
Remarks	• 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 `rangeval`. • `smuX.source.rangeX` 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 Model 2601/2602 6.0V range, or sourcing 5.0V on the Model 2611/2612/2635/2636 2.0V range.
Details	See "Range" in Section 6.
Also see	smuX.source.autorangeY
Example	Selects 1V source range for Model 2601/2602 SMU A: `smua.source.rangev = 1`

status function and attributes

The following provides a brief overview of the status model. Details on the status model are provided in Appendix D of this manual.

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, which in turn sets another bit in a lower-level register, and ultimately sets the summary bit of the Status Byte Register.

Negative and positive transition registers

- **Negative-transition register (NTR)** – When a bit in an NTR register is set by the user, the corresponding bit in the event register will set when the corresponding bit in the condition register transitions from 1 to 0.
- **Positive-transition register (PTR)** – When a bit in a PTR register is set by the user, the corresponding bit in the event register will set when the corresponding bit in the condition register transitions from 0 to 1.

Status byte and SRQ

The Status Byte Register receives the summary bits of the 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.condition	
Attribute	Status byte register.
Usage	Reads the status byte register: statbyte = status.condition

Remarks	• This attribute is used to read the status byte, which is returned as a numeric value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 7.
	• For example, assume value 129 is returned for the enable register. The binary equivalent is 10000001. This value indicates that bit B0 (MSB) and bit B7 (OSB) are set.
	• 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 summary 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 summary 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.
	  • When using `status.condition` or the *STB? common command 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.

Details	See "Status byte and service request (SRQ)" in Appendix D.

Example	Reads the status byte:
	`statbyte = status.condition`
	`print(statbyte)`
	Output: `1.29000e+02`
	The above output indicates that bits B0 (MSS) and B7 (OSB) are set.

status.measurement.*
status.measurement.condition
status.measurement.enable
status.measurement.event
status.measurement.ntr
status.measurement.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Measurement event status register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`measreg = status.measurement.condition`
	`measreg = status.measurement.current_limit.enable`
	`measreg = status.measurement.current_limit.event`
	`measreg = status.measurement.current_limit.ntr`
	`measreg = status.measurement.current_limit.ptr`
	Writes to enable, NTR and PTR registers:
	`status.measurement.enable = measreg`
	`status.measurement.ntr = measreg`
	`status.measurement.ptr = measreg`
	Set `measreg` to one of the following values:
	`0` Clears all bits.
	`status.measurement.VOLTAGE_LIMIT` Sets VLMT bit (B0).
	`status.measurement.VLMT` Sets VLMT bit (B0).
	`status.measurement.CURRENT_LIMIT` Sets ILMT bit (B1).
	`status.measurement.ILMT` Sets ILMT bit (B1).
	`status.measurement.READING_OVERFLOW` Sets ROF bit (B7).
	`status.measurement.ROF` Sets ROF bit (B7).
	`status.measurement.BUFFER_AVAILABLE` Sets BAV bit (B8).
	`status.measurement.BAV` Sets BAV bit (B8).
	`status.measurement.OUTPUT_ENABLE` Sets OE bit (B11).
	`status.measurement.OE` Sets OE bit (B11).
	`status.measurement.INSTRUMENT_SUMMARY` Sets INST bit (B13).
	`status.measurement.INST` Sets INST bit (B13).
	`measreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B0 (VLMT), set `measreg` to 1 (2^0).
	To set bit B1 (ILMT), set `measreg` to 2 (2^1).
	To set bit B8 (BAV), set `measreg` to 256 (2^8).
	To set more than one bit of the register, set `measreg` to the sum of their decimal weights. For example, to set bits B0 and B8, set `measreg` to 257 (1 + 256).

Remarks	• This attribute is used to read or write to the measurement registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 257 is returned for the register. The binary equivalent is 0000000100000001. This value indicates that bit B1(VLMT) and bit B8 (BAV) are set. • The used bits of the measurement registers are described as follows: • **Bit B0, VLMT** – Set bit indicates that the voltage limit was exceeded. This bit will be updated only when a measurement is taken or `smuX.source.compliance` is invoked. • **Bit B1, ILMT** – Set bit indicates that the current limit was exceeded. This bit will be updated only when a measurement is taken or `smuX.source.compliance` is invoked. • **Bit B7, ROF** – Set bit indicates that an overflow reading has been detected. • **Bit B8, BAV** – Set bit indicates that there is at least one reading stored in either or both of the non-volatile reading buffers. • **Bit B11, OE** – Set bit indicates that output enable has been asserted. • **Bit B13, INST** – Set bit indicates that a bit in the measurement instrument summary register is set.
Details	See "Measurement Event Registers" in Appendix D.

Example	Sets the BAV bit of the measurement enable register: `status.measurement.enable = status.measurement.BAV`

status.measurement.buffer_available.* status.measurement.buffer_available.condition status.measurement.buffer_available.enable status.measurement.buffer_available.event status.measurement.buffer_available.ntr status.measurement.buffer_available.ptr * = condition, enable, event, ntr or ptr	
Attribute	Measurement buffer available event register set.

Usage	`Reads condition, enable, event, NTR and PTR registers:`
	`measreg = status.measurement.buffer_available.condition`
	`measreg = status.measurement.buffer_available.enable`
	`measreg = status.measurement.buffer_available.event`
	`measreg = status.measurement.buffer_available.ntr`
	`measreg = status.measurement.buffer_available.ptr`
	Writes to enable, NTR and PTR registers:
	`status.measurement.buffer_available.enable = measreg`
	`status.measurement.buffer_available.ntr = measreg`
	`status.measurement.buffer_available.ptr = measreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.measurement.buffer_available.SMUA` Sets SMUA bit (B1).
	`status.measurement.buffer_available.SMUB` Sets SMUB (B2).
	`measreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `measreg` to 2 (2^1).
	To set bit B2 (SMUB), set `measreg` to 4 (2^2).
	To set both bits, set `measreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `measreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the measurement buffer available registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1(SMUA) and bit B2 (SMUB) are set. • The used bits of the measurement buffer available registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled BAV bit for the SMU A measurement register is set. • **Bit B2, SMUB** – Set bit indicates the enabled BAV bit for the SMU B measurement register is set.
Details	See "Measurement Event Registers" in Appendix D.

Example	Sets the SMUA bit of the measurement buffer available enable register:
	`status.measurement.buffer_available.enable =`
	`status.measurement.buffer_available.SMUA`

status.measurement.current_limit.*
status.measurement.current_limit.condition
status.measurement.current_limit.enable
status.measurement.current_limit.event
status.measurement.current_limit.ntr
status.measurement.current_limit.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Measurement current limit event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`measreg = status.measurement.current_limit.condition`
	`measreg = status.measurement.current_limit.enable`
	`measreg = status.measurement.current_limit.event`
	`measreg = status.measurement.current_limit.ntr`
	`measreg = status.measurement.current_limit.ptr`
	Writes to enable, NTR and PTR registers:
	`status.measurement.current_limit.enable = measreg`
	`status.measurement.current_limit.ntr = measreg`
	`status.measurement.current_limit.ptr = measreg`
	Set `measreg` to one of the following values:
	`0` Clears all bits.
	`status.measurement.current_limit.SMUA` Sets SMUA bit (B1).
	`status.measurement.current_limit.SMUB` Sets SMUB (B2).
	`measreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `measreg` to 2 (2^1).
	To set bit B2 (SMUB), set `measreg` to 4 (2^2).
	To set both bits, set `measreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `measreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the measurement current limit registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1(SMUA) and bit B2 (SMUB) are set. • The used bits of the measurement current limit registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled ILMT bit for the SMU A measurement register is set. • **Bit B2, SMUB** – Set bit indicates the enabled ILMT bit for the SMU B measurement register is set.
Details	See "Measurement Event Registers" in Appendix D.
Example	Sets the SMUA bit of the measurement current limit enable register: `status.measurement.current_limit.enable = status.measurement.current_limit.SMUA`

status.measurement.instrument.*
status.measurement.instrument.condition
status.measurement.instrument.enable
status.measurement.instrument.event
status.measurement.instrument.ntr
status.measurement.instrument.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Measurement instrument event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`measreg = status.measurement.instrument.condition`
	`measreg = status.measurement.instrument.enable`
	`measreg = status.measurement.instrument.event`
	`measreg = status.measurement.instrument.ntr`
	`measreg = status.measurement.instrument.ptr`
	Writes to enable, NTR and PTR registers:
	`status.measurement.instrument.enable = measreg`
	`status.measurement.instrument.ntr = measreg`
	`status.measurement.instrument.ptr = measreg`
	Set `measreg` to one of the following values:
	`0` Clears all bits.
	`status.measurement.instrument.SMUA` Sets SMUA bit (B1).
	`status.measurement.instrument.SMUB` Sets SMUB (B2).
	`measreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `measreg` to 2 (2^1).
	To set bit B2 (SMUB), set `measreg` to 4 (2^2).
	To set both bits, set `measreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `measreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the measurement instrument registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1 (SMUA) and bit B2 (SMUB) are set. • The used bits of the measurement instrument registers are described as follows: • **Bit B1, SMUA** – Set bit indicates one or more enabled bits for the SMU A measurement register is set. • **Bit B2, SMUB** – Set bit indicates one or more enabled bits for the SMU B measurement register is set.
Details	See "Measurement Event Registers" in Appendix D.
Example	Sets the SMUA bit of the measurement instrument enable register: `status.measurement.instrument.enable = status.measurement.instrument.SMUA`

status.measurement.instrument.smuX.*
status.measurement.instrument.smuX.condition
status.measurement.instrument.smuX.enable
status.measurement.instrument.smuX.event
status.measurement.instrument.smuX.ntr
status.measurement.instrument.smuX.ptr
*** = condition, enable, event, ntr or ptr**

smuX = smua or smub

Attribute	Measurement SMU event status register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`measreg = status.measurement.instrument.smuX.condition`
	`measreg = status.measurement.instrument.smuX.enable`
	`measreg = status.measurement.instrument.smuX.event`
	`measreg = status.measurement.instrument.smuX.ntr`
	`measreg = status.measurement.instrument.smuX.ptr`
	Writes to enable, NTR and PTR registers:
	`status.measurement.instrument.smuX.enable = measreg`
	`status.measurement.instrument.smuX.ntr = measreg`
	`status.measurement.instrument.smuX.ptr = measreg`
	Set `measreg` to one of the following values:
	`0` Clears all bits.
	`status.measurement.VOLTAGE_LIMIT` Sets VLMT bit (B0).
	`status.measurement.VLMT` Sets VLMT bit (B0).
	`status.measurement.CURRENT_LIMIT` Sets ILMT bit (B1).
	`status.measurement.ILMT` Sets ILMT bit (B1).
	`status.measurement.READING_OVERFLOW` Sets ROF (B7).
	`status.measurement.ROF` Sets ROF (B7).
	`status.measurement.BUFFER_AVAILABLE` Sets BAV (B8).
	`status.measurement.BAV` Sets BAV (B8).
	`measreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B0 (VLMT), set `measreg` to 1 (2^0).
	To set bit B1 (ILMT), set `measreg` to 2 (2^1).
	To set bit B8 (BAV), set `measreg` to 256 (2^8).
	To set more than one bit of the register, set `measreg` to the sum of their decimal weights.
	For example, to set bits B0 and B8, set `measreg` to 257 (1 + 256).

Remarks	• This attribute is used to read or write to the measurement SMU event registers.
	• Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15.
	• For example, assume value 257 is returned for the enable register. The binary equivalent is 0000000100000001. This value indicates that bit B0 (VLMT) and bit B8 (BAV) are set.
	• The used bits of the measurement SMU event registers are described as follows:
	• **Bit B0, VLMT** – Set bit indicates that the voltage limit was exceeded. This bit will be updated only when a measurement is taken or `smuX.source.compliance` is invoked.
	• **Bit B1, ILMT** – Set bit indicates that the current limit was exceeded. This bit will be updated only when a measurement is taken or `smuX.source.compliance` is invoked.
	  • **Bit B7, ROF –** Set bit indicates that an overflow reading has been detected.
	  • **Bit B8, BAV –** Set bit indicates that there is at least one reading stored in either or both of the non-volatile reading buffers.

Details	See "Measurement Event Registers" in Appendix D.
Example	Sets the BAV bit of the measurement SMU A enable register: `status.measurement.instrument.smuX.enable = status.measurement.BAV`

status.measurement.reading_overflow.*
status.measurement.reading_overflow.condition
status.measurement.reading_overflow.enable
status.measurement.reading_overflow.event
status.measurement.reading_overflow.ntr
status.measurement.reading_overflow.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Measurement reading overflow event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`measreg = status.measurement.reading_overflow.condition`
	`measreg = status.measurement.reading_overflow.enable`
	`measreg = status.measurement.reading_overflow.event`
	`measreg = status.measurement.reading_overflow.ntr`
	`measreg = status.measurement.reading_overflow.ptr`
	Writes to enable, NTR and PTR registers:
	`status.measurement.reading_overflow.enable = measreg`
	`status.measurement.reading_overflow.ntr = measreg`
	`status.measurement.reading_overflow.ptr = measreg`
	Set `measreg` to one of the following values:
	`0` Clears all bits.
	`status.measurement.reading_overflow.SMUA` Sets SMUA bit (B1).
	`status.measurement.reading_overflow.SMUB` Sets SMUB (B2).
	`measreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `measreg` to 2 (2^1).
	To set bit B2 (SMUB), set `measreg` to 4 (2^2).
	To set both bits, set `measreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `measreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the measurement current limit registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1(SMUA) and bit B2 (SMUB) are set. • The used bits of the measurement current limit registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled ILMT bit for the SMU A measurement register is set. • **Bit B2, SMUB** – Set bit indicates the enabled ILMT bit for the SMU B measurement register is set.
Details	See "Measurement Event Registers" in Appendix D.
Example	Sets the SMUA bit of the measurement reading overflow enable register: `status.measurement.reading_overflow.enable = status.measurement.reading_overflow.SMUA`

status.measurement.voltage_limit.*
status.measurement.voltage_limit.condition
status.measurement.voltage_limit.enable
status.measurement.voltage_limit.event
status.measurement.voltage_limit.ntr
status.measurement.voltage_limit.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Measurement voltage limit event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`measreg = status.measurement.voltage_limit.condition`
	`measreg = status.measurement.voltage_limit.enable`
	`measreg = status.measurement.voltage_limit.event`
	`measreg = status.measurement.voltage_limit.ntr`
	`measreg = status.measurement.voltage_limit.ptr`
	Writes to enable, NTR and PTR registers:
	`status.measurement.voltage_limit.enable = measreg`
	`status.measurement.voltage_limit.ntr = measreg`
	`status.measurement.voltage_limit.ptr = measreg`
	Set `measreg` to one of the following values:
	`0` Clears all bits.
	`status.measurement.voltage_limit.SMUA` Sets SMUA bit (B1).
	`status.measurement.voltage_limit.SMUB` Sets SMUB (B2).
	`measreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `measreg` to 2 (2^1).
	To set bit B2 (SMUB), set `measreg` to 4 (2^2).
	To set both bits, set `measreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `measreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the measurement voltage limit registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1(SMUA) and bit B2 (SMUB) are set. • The used bits of the measurement voltage limit registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled VLMT bit for the SMU A measurement register is set. • **Bit B2, SMUB** – Set bit indicates the enabled VLMT bit for the SMU B measurement register is set.
Details	See "Measurement Event Registers" in Appendix D.
Example	Sets the SMUA bit of the measurement voltage limit enable register: `status.measurement.voltage_limit.enable = status.measurement.voltage_limit.SMUA`

status.node_enable	
Attribute	Status node enable register.

Usage	Reads status node enable register:
	`nodeenabreg = status.node_enable`
	`Writes to system enable register:`
	`status.node_enable = nodeenabreg`
	Set `nodeenabreg` to one of the following values:
	`0` Clears all bits.
	`status.MEASUREMENT_SUMMARY_BIT` Sets (enables) MSB bit (B0).
	`status.MSB` Sets (enables) MSB bit (B0).
	`status.ERROR_AVAILABLE` Sets (enables) EAV bit (B2).
	`status.EAV` Sets (enables) EAV bit (B2).
	`status.QUESTIONABLE_SUMMARY_BIT` Sets (enables) QSB bit (B3).
	`status.QSB` Sets (enables) QSB bit (B3).
	`status.MESSAGE_AVAILABLE` Sets (enables) MAV bit (B4).
	`status.MAV` Sets (enables) MAV bit (B4).
	`status.EVENT_SUMMARY_BIT` Sets (enables) ESB bit (B5).
	`status.ESB` Sets (enables) ESB bit (B5).
	`status.MASTER_SUMMARY_STATUS` Sets (enables) MSS bit (B6).
	`status.MSS` Sets (enables) MSS bit (B6).
	`status.OPERATION_SUMMARY_BIT` Sets (enables) OSB bit (B7).
	`status.OSB` Sets (enables) OSB bit (B7).
	`nodeenabreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B0 (MSB), set `nodeenabreg` to 1 (2^0).
	To set bit B2 (EAV), set `nodeenabreg` to 4 (2^2).
	To set bit B7 (OSB), set `nodeenabreg` to 128 (2^7).
	To set more than one bit of the register, set `nodeenabreg` to the sum of their decimal weights. For example, to set bits B0 and B7, set `nodeenabreg` to 129 (1 + 128).

Remarks	• This attribute is used to read or write to the status node enable register. • Reading the node enable status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 7. • For example, assume value 129 is returned for the node enable register. The binary equivalent is 10000001. This value indicates that bit B0 (MSB) and bit B7 (OSB) are set. • Assigning a value to this attribute enables one or more status events for enabled system nodes. When an enabled status event occurs, it will set one or more enabled system node bits of the system registers (see "status.system.*" registers). • The status node enable register uses most of the same summary events as the status byte. Bit B1(MSB) is not used, and bit B6 is used as Master Summary Status (MSS). For details, see "status.condition" register.
Details	See "Status byte and service request (SRQ)" in Appendix D.
Example	Sets the MSB bit of the status node enable register: `status.node_enable = status.MSB`

status.node_event	
Attribute	Status node event register.
Usage	Reads the status node event register: `nodeeventreg = status.node_event`
Remarks	• This attribute is used to read the status node event register, which is returned as a numeric value. Reading this register returns a value. The binary equivalent of the returned value. The least significant bit of the binary number is bit 0, and the most significant bit is bit 7. • For example, assume value 129 is returned for the event register. The binary equivalent is 10000001. This value indicates that bit B0 (MSB) and bit B7 (OSB) are set. • The status node event register uses most of the same summary events as the status byte. Bit B1(MSB) is not used, and bit B6 is used as Master Summary Status (MSS). For details, see "status.condition" register.
Details	See "Status byte and service request (SRQ)" in Appendix D.
Example	Reads the status node event register: nodeeventreg `= status.`node_event `print(`nodeeventreg`)` Output: `1.29000e+02` The above output indicates that bits B0 (MSS) and B7 (OSB) are set.

status.operation.*
status.operation.condition
status.operation.enable
status.operation.event
status.operation.ntr
status.operation.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Operation event status register set.
Usage	Reads condition, enable, event, NTR and PTR registers: operreg = status.operation.condition operreg = status.operation.enable operreg = status.operation.event operreg = status.operation.ntr operreg = status.operation.ptr Writes to enable, NTR and PTR registers: status.operation.enable = operreg status.operation.ntr = operreg status.operation.ptr = operreg Set operreg to one of the following values: 0 — Clears all bits. status.operation.CALIBRATING — Sets CAL bit (B0). status.operation.CAL — Sets CAL bit (B0). status.operation.MEASURING — Sets MEAS bit (B4). status.operation.MEAS — Sets MEAS bit (B4). status.operation.PROMPTS — Sets PRMPTS bit (B11). status.operation.PRMPTS — Sets PRMPTS bit (B11). status.operation.USER — Sets USER bit (B12). status.operation.INSTRUMENT_SUMMARY — Sets INST bit (B13). status.operation.INST — Sets INST bit (B13). status.operation.PROGRAM_RUNNING — Sets PROG bit (B14). status.operation.PROG — Sets PROG bit (B14). operreg can also be set to the decimal weight of the bit to be set. Examples: To set bit B0 (CAL), set operreg to 1 (2^0). To set bit B4 (MEAS), set operreg to 16 (2^4). To set bit B11 (PRMPTS), set operreg to 2048 (2^{11}). To set more than one bit of the register, set operreg to the sum of their decimal weights. For example, to set bits B0 and B4, set operreg to 17 (1 + 16).

Remarks	• This attribute is used to read or write to the operation event registers.
	• Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15.
	• For example, assume value 17 is returned for the enable register. The binary equivalent is 0000000000010001. This value indicates that bit B0 (CAL) and bit B4 (MEAS) are set.
	• The used bits of the operation event registers are described as follows:
	• **Bit B0, CAL** – Set bit indicates that one or more channels are calibrating.
	  • **Bit B4, MEAS –** Bit will be set when taking an overlapped measurement, but it will not set when taking a normal synchronous measurement.
	  • **Bit B11, PRMPTS –** Set bit indicates that command prompts are enabled.
	  • **Bit B12, USER –** Set bit indicates that an enabled bit in the operation status user register is set.
	  • **Bit B13, INST –** Set bit indicates that an enabled bit in the operation status instrument summary register is set.
	  • **Bit B14, PROG** – Set bit indicates that a program is running.

Details	See "Operation Event Registers" in Appendix D.
Example	Sets the MEAS bit of the operation enable register: status.operation.enable = status.operation.MEAS

status.operation.calibrating.*
status.operation.calibrating.condition
status.operation.calibrating.enable
status.operation.calibrating.event
status.operation.calibrating.ntr
status.operation.calibrating.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Operation calibration event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`operreg = status.operation.calibrating.condition`
	`operreg = status.operation.calibrating.enable`
	`operreg = status.operation.calibrating.event`
	`operreg = status.operation.calibrating.ntr`
	`operreg = status.operation.calibrating.ptr`
	Writes to enable, NTR and PTR registers:
	`status.operation.calibrating.enable = operreg`
	`status.operation.calibrating.ntr = operreg`
	`status.operation.calibrating.ptr = operreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.operation.calibrating.SMUA` Sets SMUA bit (B1).
	`status.operation.calibrating.SMUB` Sets SMUB (B2).
	`operreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `operreg` to 2 (2^1).
	To set bit B2 (SMUB), set `operreg` to 4 (2^2).
	To set both bits, set `operreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `operreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the operation calibration registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1(SMUA) and bit B2 (SMUB) are set. • The used bits of the operation calibration registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled CAL bit for the SMU A operation register is set. • **Bit B2, SMUB** – Set bit indicates the enabled CAL bit for the SMU B operation register is set.
Details	See "Operation Event Registers" in Appendix D.
Example	Sets the SMUA bit of the operation calibration enable register: `status.operation.calibrating.enable = status.operation.calibrating.SMUA`

status.operation.instrument.*
status.operation.instrument.condition
status.operation.instrument.enable
status.operation.instrument.event
status.operation.instrument.ntr
status.operation.instrument.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Operation instrument event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`operreg = status.operation.instrument.condition`
	`operreg = status.operation.instrument.enable`
	`operreg = status.operation.instrument.event`
	`operreg = status.operation.instrument.ntr`
	`operreg = status.operation.instrument.ptr`
	Writes to enable, NTR and PTR registers:
	`status.operation.instrument.enable = operreg`
	`status.operation.instrument.ntr = operreg`
	`status.operation.instrument.ptr = operreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.operation.instrument.SMUA` Sets SMUA bit (B1).
	`status.operation.instrument.SMUB` Sets SMUB (B2).
	`operreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `operreg` to 2 (2^1).
	To set bit B2 (SMUB), set `operreg` to 4 (2^2).
	To set both bits, set `operreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `operreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the operation instrument registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1 (SMUA) and bit B2 (SMUB) are set. • The used bits of the operation instrument registers are described as follows: • **Bit B1, SMUA** – Set bit indicates one or more enabled bits for the SMU A operation register is set. • **Bit B2, SMUB** – Set bit indicates one or more enabled bits for the SMU B operation register is set.
Details	See "Operation Event Registers" in Appendix D.
Example	Sets the SMUA bit of the operation instrument enable register: `status.operation.instrument.enable = status.operation.instrument.SMUA`

status.operation.instrument.*
status.operation.instrument.smuX.condition
status.operation.instrument.smuX.enable
status.operation.instrument.smuX.event
status.operation.instrument.smuX.ntr
status.operation.instrument.smuX.ptr
*** = condition, enable, event, ntr or ptr**

smuX = smua or smub

Attribute	Operation SMU event register sets.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`operreg = status.operation.instrument.smuX.condition`
	`operreg = status.operation.instrument.smuX.enable`
	`operreg = status.operation.instrument.smuX.event`
	`operreg = status.operation.instrument.smuX.ntr`
	`operreg = status.operation.instrument.smuX.ptr`
	Writes to enable, NTR and PTR registers:
	`status.operation.instrument.smuX.enable = operreg`
	`status.operation.instrument.smuX.ntr = operreg`
	`status.operation.instrument.smuX.ptr = operreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.operation.CALIBRATING` Sets CAL bit (B0).
	`status.operation.CAL` Sets CAL bit (B0).
	`status.operation.MEASURING` Sets MEAS bit (B4).
	`status.operation.MEAS` Sets MEAS bit (B4).
	`status.operation.PROMPTS` Sets PRMPTS bit (B11).
	`status.operation.PRMPTS` Sets PRMPTS bit (B11).
	`status.operation.USER` Sets USER bit (B12).
	`status.operation.PROGRAM_RUNNING` Sets PROG bit (B14).
	`status.operation.PROG` Sets PROG bit (B14).
	`operreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B0 (CAL), set `operreg` to 1 (2^0).
	To set bit B4 (MEAS), set `operreg` to 16 (2^4).
	To set bit B11 (PRMPTS), set `operreg` to 2048 (2^{11}).
	To set more than one bit of the register, set `operreg` to the sum of their decimal weights. For example, to set bits B0 and B4, set `operreg` to 17 (1 + 16).

Remarks	• This attribute is used to read or write to the operation SMU event registers.
	• Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15.
	• For example, assume value 17 is returned for the enable register. The binary equivalent is 0000000000010001. This value indicates that bit B0 (CAL) and bit B4 (MEAS) are set.
	• The used bits of the operation SMU event registers are described as follows:
	• **Bit B0, CAL** – Set bit indicates that one or more channels are calibrating.
	  • **Bit B4, MEAS –** Bit will be set when taking an overlapped measurement, but it will not set when taking a normal synchronous measurement.
	  • **Bit B11, PRMPTS –** Set bit indicates that command prompts are enabled.
	  • **Bit B12, USER –** Set bit indicates that an enabled bit in the operation status user register is set.
	  • **Bit B14, PROG** – Set bit indicates that a program is running.

Details	See "Operation Event Registers" in Appendix D.
Example	Sets the MEAS bit of the operation SMU A enable register: `status.operation.instrument.smua.enable = status.operation.MEAS`

status.operation.measuring.*
status.operation.measuring.condition
status.operation.measuring.enable
status.operation.measuring.event
status.operation.measuring.ntr
status.operation.measuring.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Operation measurement event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`operreg = status.operation.measuring.condition`
	`operreg = status.operation.measuring.enable`
	`operreg = status.operation.measuring.event`
	`operreg = status.operation.measuring.ntr`
	`operreg = status.operation.measuring.ptr`
	Writes to enable, NTR and PTR registers:
	`status.operation.measuring.enable = operreg`
	`status.operation.measuring.ntr = operreg`
	`status.operation.measuring.ptr = operreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.operation.measuring.SMUA` Sets SMUA bit (B1).
	`status.operation.measuring.SMUB` Sets SMUB (B2).
	`operreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `operreg` to 2 (2^1).
	To set bit B2 (SMUB), set `operreg` to 4 (2^2).
	To set both bits, set `operreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `operreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the operation measurement registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1(SMUA) and bit B2 (SMUB) are set. • The used bits of the operation measurement registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled MEAS bit for the SMU A operation register is set. • **Bit B2, SMUB** – Set bit indicates the enabled MEAS bit for the SMU B operation register is set.
Details	See "Operation Event Registers" in Appendix D.

Example	Sets the SMUA bit of the operation measurement enable register:
	`status.operation.measuring.enable =`
	`status.measuring.calibrating.SMUA`

status.operation.user.*
status.operation.user.condition
status.operation.user.enable
status.operation.user.event
status.operation.user.ntr
status.operation.user.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Operation user event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`operreg = status.operation.user.condition`
	`operreg = status.operation.user.enable`
	`operreg = status.operation.user.event`
	`operreg = status.operation.user.ntr`
	`operreg = status.operation.user.ptr`
	Writes to condition, enable, NTR and PTR registers:
	`status.operation.user.condition = operreg`
	`status.operation.user.enable = operreg`
	`status.operation.user.ntr = operreg`
	`status.operation.user.ptr = operreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.operation.user.BIT0` Sets user BIT0.
	`status.operation.user.BIT1` Sets user BIT1.
	`status.operation.user.BIT2` Sets user BIT2.
	`status.operation.user.BIT3` Sets user BIT3.
	`status.operation.user.BIT4` Sets user BIT4.
	`status.operation.user.BIT5` Sets user BIT5.
	`status.operation.user.BIT6` Sets user BIT6.
	`status.operation.user.BIT7` Sets user BIT.
	`status.operation.user.BIT8` Sets user BIT.
	`status.operation.user.BIT9` Sets user BIT.
	`status.operation.user.BIT10` Sets user BIT.
	`status.operation.user.BIT11` Sets user BIT.
	`status.operation.user.BIT12` Sets user BIT.
	`status.operation.user.BIT13` Sets user BIT.
	`status.operation.user.BIT14` Sets user BIT.
	`operreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set user BIT0, set `operreg` to 1 (2^0).
	To set user BIT4, set `operreg` to 16 (2^4).
	To set user BIT11, set `operreg` to 2048 (2^{11}).
	To set more than one bit of the register, set `operreg` to the sum of their decimal weights. For example, to set BIT0 and BIT4, set `operreg` to 17 (1 + 16).

Remarks	• This attribute is used to read or write to the operation user registers. • Bits of the user event register are set by setting the corresponding bits of the user enable register and the user condition register. For example, the following code will set B1 (Bit 1) of the user event register: • `status.operation.user.enable = 2` • `status.operation.user.condition = 2` • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 17 is returned for the enable register. The binary equivalent is 0000000000010001. This value indicates that BIT0 and BIT4 are set.
Details	See "Operation Event Registers" in Appendix D.
Example	Sets user BIT0 of the operation user enable register: `status.operation.user.enable = status.operation.user.BIT0`

status.questionable.*
status.questionable.condition
status.questionable.enable
status.questionable.event
status.questionable.ntr
status.questionable.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Questionable event status register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`quesreg = status.questionable.condition`
	`quesreg = status.questionable.enable`
	`quesreg = status.questionable.event`
	`quesreg = status.questionable.ntr`
	`quesreg = status.questionable.ptr`
	Writes to enable, NTR and PTR registers:
	`status.questionable.enable = quesreg`
	`status.questionable.ntr = quesreg`
	`status.questionable.ptr = quesreg`
	Set `quesreg` to one of the following values:
	`0` Clears all bits.
	`status.questionable.CALIBRATING` Sets CAL bit (B8).
	`status.questionable.CAL` Sets CAL bit (B8).
	`status.questionable.UNSTABLE_OUTPUT` Sets UO bit (B9).
	`status.questionable.UO` Sets UO bit (B9).
	`status.questionable.OVER_TEMPERATURE` Sets OTEMP bit (B12).
	`status.questionable.OTEMP` Sets OTEMP bit (B12).
	`status.questionable.INSTRUMENT_SUMMARY` Sets INST bit (B13).
	`status.questionable.INST` Sets INST bit (B13).
	`quesreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B8 (CAL), set `quesreg` to 256 (2^8).
	To set bit B9 (UO), set `quesreg` to 512 (2^9).
	To set bit B12 (OTEMP), set `quesreg` to 4096 (2^{12}).
	To set bit B13 (INST), set `quesreg` to 8192 (2^{13}).
	To set more than one bit of the register, set `quesreg` to the sum of their decimal weights.
	For example, to set bits B8 and B12, set `quesreg` to 4352 (256 + 4096).

Remarks	• This attribute is used to read or write to the questionable event registers.
	• Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15.
	• For example, assume value 4352 is returned for the enable register. The binary equivalent is 0001000100000000. This value indicates that bit B8 (CAL) and bit B12 (OTEMP) are set.
	• The used bits of the questionable event registers are described as follows:
	  • **Bit B8, CAL** – Set bit indicates that calibration is questionable.
	  • **Bit B9, UO** – Set bit indicates that an unstable output condition was detected.
	  • **Bit B12, OTEMP** – Set bit indicates that an over temperature condition was detected.
	  • **Bit B13, INST** – Set bit indicates that a bit in the questionable instrument summary register is set.

Details	See "`status.operation.user.condition = 2`" in Appendix D.
Example	Sets the OTEMP bit of the questionable enable register: `status.questionable.enable = status.questionable.OTEMP`

status.questionable.calibration.*
status.questionable.calibration.condition
status.questionable.calibration.enable
status.questionable.calibration.event
status.questionable.calibration.ntr
status.questionable.calibration.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Questionable calibration event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`quesreg = status.questionable.calibration.condition`
	`quesreg = status.questionable.calibration.enable`
	`quesreg = status.questionable.calibration.event`
	`quesreg = status.questionable.calibration.ntr`
	`quesreg = status.questionable.calibration.ptr`
	Writes to enable, NTR and PTR registers:
	`status.questionable.calibration.enable = quesreg`
	`status.questionable.calibration.ntr = quesreg`
	`status.questionable.calibration.ptr = quesreg`
	Set `quesreg` to one of the following values:
	`0` Clears all bits.
	`status.questionable.calibration.SMUA` Sets SMUA bit (B1).
	`status.questionable.calibration.SMUB` Sets SMUB (B2).
	`quesreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `quesreg` to 2 (2^1).
	To set bit B2 (SMUB), set `quesreg` to 4 (2^2).
	To set both bits, set `quesreg` to the sum of the decimal weights of both bits. To set bits B1 and B2, set `quesreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the questionable calibration registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1(SMUA) and bit B2 (SMUB) are set. • The used bits of the questionable calibration registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled CAL bit for the SMU A questionable register is set. • **Bit B2, SMUB** – Set bit indicates the enabled CAL bit for the SMU B questionable register is set.
Details	See "`status.operation.user.condition = 2`" in Appendix D.
Example	Sets the SMUA bit of the questionable calibration enable register: `status.questionable.calibration.enable = status.questionable.calibration.SMUA`

status.questionable.instrument.*
status.questionable.instrument.condition
status.questionable.instrument.enable
status.questionable.instrument.event
status.questionable.instrument.ntr
status.questionable.instrument.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Questionable instrument event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`quesreg = status.questionable.instrument.condition`
	`quesreg = status.questionable.instrument.enable`
	`quesreg = status.questionable.instrument.event`
	`quesreg = status.questionable.instrument.ntr`
	`quesreg = status.questionable.instrument.ptr`
	Writes to enable, NTR and PTR registers:
	`status.questionable.instrument.enable = quesreg`
	`status.questionable.instrument.ntr = quesreg`
	`status.questionable.instrument.ptr = quesreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.questionable.instrument.SMUA` Sets SMUA bit (B1).
	`status.questionable.instrument.SMUB` Sets SMUB (B2).
	`quesreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `quesreg` to 2 (2^1).
	To set bit B2 (SMUB), set `quesreg` to 4 (2^2).
	To set both bits, set `quesreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `quesreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the questionable instrument registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1 (SMUA) and bit B2 (SMUB) are set. • The used bits of the questionable instrument registers are described as follows: • **Bit B1, SMUA** – Set bit indicates one or more enabled bits for the SMU A questionable register is set. • **Bit B2, SMUB** – Set bit indicates one or more enabled bits for the SMU B questionable register is set.
Details	See "`status.operation.user.condition = 2`" in Appendix D.
Example	Sets the SMUA bit of the questionable instrument enable register: `status.questionable.instrument.enable = status.questionable.instrument.SMUA`

status.questionable.instrument.smuX.*
status.questionable.instrument.smuX.condition
status.questionable.instrument.smuX.enable
status.questionable.instrument.smuX.event
status.questionable.instrument.smuX.ntr
status.questionable.instrument.smuX.ptr
*** = condition, enable, event, ntr or ptr**

smuX = smua or smub

Attribute	Questionable SMU event register sets.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`quesreg = status.questionable.instrument.smuX.condition`
	`quesreg = status.questionable.instrument.smuX.enable`
	`quesreg = status.questionable.instrument.smuX.event`
	`quesreg = status.questionable.instrument.smuX.ntr`
	`quesreg = status.questionable.instrument.smuX.ptr`
	Writes to enable, NTR and PTR registers:
	`status.questionable.instrument.smuX.enable = quesreg`
	`status.questionable.instrument.smuX.ntr = quesreg`
	`status.questionable.instrument.smuX.ptr = quesreg`
	Set `quesreg` to one of the following values:
	`0` Clears all bits.
	`status.questionable.CALIBRATING` Sets CAL bit (B8).
	`status.questionable.CAL` Sets CAL bit (B8).
	`status.questionable.OVER_TEMPERATURE` Sets OTEMP bit (B12).
	`status.questionable.OTEMP` Sets OTEMP bit (B12).
	`status.questionable.UNSTABLE_OUTPUT` Sets UO bit (B9)
	`status.questionable.UO` Sets UO bits (B9)
	`quesreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B8 (CAL), set `quesreg` to 256 (2^8).
	To set bit B9 (UO), set quesreg to 512 (2^9).
	To set bit B12 (OTEMP), set `quesreg` to 4096 (2^{12}).
	To set both bits of the register, set `quesreg` to the sum of the decimal weights.
	To set bits B8 and B12, set `quesreg` to 4362 (256 + 4096).

Remarks	• This attribute is used to read or write to the questionable SMU event registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 4362 is returned for the enable register. The binary equivalent is 0001000100000000. This value indicates that bit B8 (Cal bit) and bit B4 (Meas bit) are set. • The used bits of the questionable SMU event registers are described as follows: • **Bit B8, CAL** – Set bit indicates that calibration is questionable. Bit B9, UO - Set bit indicates that an unstable output condition was detected. • **Bit B12, OTEMP –** Set bit indicates that an over temperature condition was detected.
Details	See "`status.operation.user.condition = 2`" in Appendix D.
Example	Sets the OTEMP bit of the questionable SMU A enable register: `status.questionable.instrument.smua.enable = status.questionable.OTEMP`

status.questionable.over_temperature.*
status.questionable.over_temperature.condition
status.questionable.over_temperature.enable
status.questionable.over_temperature.event
status.questionable.over_temperature.ntr
status.questionable.over_temperature.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Questionable over temperature event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`quesreg = status.questionable.over_temperature.condition`
	`quesreg = status.questionable.over_temperature.enable`
	`quesreg = status.questionable.over_temperature.event`
	`quesreg = status.questionable.over_temperature.ntr`
	`quesreg = status.questionable.over_temperature.ptr`
	Writes to enable, NTR and PTR registers:
	`status.questionable.over_temperature.enable = quesreg`
	`status.questionable.over_temperature.ntr = quesreg`
	`status.questionable.over_temperature.ptr = quesreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.questionable.over_temperature.SMUA` Sets SMUA bit (B1).
	`status.questionable.over_temperature.SMUB` Sets SMUB (B2).
	`quesreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `quesreg` to 2 (2^1).
	To set bit B2 (SMUB), set `quesreg` to 4 (2^2).
	To set both bits, set `quesreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `quesreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the questionable over temperature registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1 (SMUA) and bit B2 (SMUB) are set. • The used bits of the questionable instrument registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled OTEMP bit for the SMU A questionable register is set. • **Bit B2, SMUB** – Set bit indicates the enabled OTEMP bit for the SMU B questionable register is set.
Details	See "`status.operation.user.condition = 2`" in Appendix D.
Example	Sets the SMUA bit of the questionable over temperature enable register: `status.questionable.over_temperature.enable = status.questionable.over_temperature.SMUA`

status.questionable.unstable_output*
status.questionable.unstable_output.condition
status.questionable.unstable_output.enable
status.questionable.unstable_output.event
status.questionable.unstable_output.ntr
status.questionable.unstable_output.ptr
*** = condition, enable, event, ntr or ptr**

Attribute	Questionable unstable output event register set.

Usage	Reads condition, enable, event, NTR and PTR registers:
	`quesreg = status.questionable.unstable_output.condition`
	`quesreg = status.questionable.unstable_output.enable`
	`quesreg = status.questionable.unstable_output.event`
	`quesreg = status.questionable.unstable_output.ntr`
	`quesreg = status.questionable.unstable_output.ptr`
	Writes to enable, NTR and PTR registers:
	`status.questionable.unstable_output.enable = quesreg`
	`status.questionable.unstable_output.ntr = quesreg`
	`status.questionable.unstable_output.ptr = quesreg`
	Set `operreg` to one of the following values:
	`0` Clears all bits.
	`status.questionable.unstable_output.SMUA` Sets SMUA bit (B1).
	`status.questionable.unstable_output.SMUB` Sets SMUB (B2).
	`quesreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B1 (SMUA), set `quesreg` to 2 (2^1).
	To set bit B2 (SMUB), set `quesreg` to 4 (2^2).
	To set both bits, set `quesreg` to the sum of the decimal weights of both bits.
	To set bits B1 and B2, set `quesreg` to 6 (2 + 4).

Remarks	• This attribute is used to read or write to the questionable unstable output registers. • Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15. • For example, assume value 6 is returned for the enable register. The binary equivalent is 0000000000000110. This value indicates that bit B1 (SMUA) and bit B2 (SMUB) are set. • The used bits of the questionable instrument registers are described as follows: • **Bit B1, SMUA** – Set bit indicates the enabled UO bit for the SMU A questionable register is set. • **Bit B2, SMUB** – Set bit indicates the enabled UO bit for the SMU B questionable register is set.
Details	See "`status.operation.user.condition = 2`" in Appendix D.
Example	Sets the SMUA bit of the questionable unstable output enable register: `status.questionable.unstable_output.enable = status.questionable.unstable_output.SMUA`

status.request_enable	
Attribute	Service request enable register.

Usage	Reads service request enable register:
	`servenabreg = status.request_enable`
	Writes to system enable register:
	`status.request_enable = servenabreg`
	Set `servenabreg` to one of the following values:
	`0` Clears all bits.
	`status.MEASUREMENT_SUMMARY_BIT` Sets (enables) MSB bit (B0).
	`status.MSB` Sets (enables) MSB bit (B0).
	`status.SYSTEM_SUMMARY_BIT` Sets (enables) SSB bit (B1).
	`status.SSB` Sets (enables) SSB bit (B1).
	`status.ERROR_AVAILABLE` Sets (enables) EAV bit (B2).
	`status.EAV` Sets (enables) EAV bit (B2).
	`status.QUESTIONABLE_SUMMARY_BIT` Sets (enables) QSB bit (B3).
	`status.QSB` Sets (enables) QSB bit (B3).
	`status.MESSAGE_AVAILABLE` Sets (enables) MAV bit (B4).
	`status.MAV` Sets (enables) MAV bit (B4).
	`status.EVENT_SUMMARY_BIT` Sets (enables) ESB bit (B5).
	`status.ESB` Sets (enables) ESB bit (B5).
	`status.OPERATION_SUMMARY_BIT` Sets (enables) OSB bit (B7).
	`status.OSB` Sets (enables) OSB bit (B7).
	`servenabreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B0 (MSB), set `servenabreg` to 1 (2^0).
	To set bit B2 (EAV), set `servenabreg` to 4 (2^2).
	To set bit B7 (OSB), set `servenabreg` to 128 (2^7).
	To set more than one bit of the register, set `servenabreg` to the sum of their decimal weights. For example, to set bits B0 and B7, set `servenabreg` to 129 (1 + 128).

Remarks	• This attribute is used to read or write to the service request enable register. • Reading the service request enable status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 7. • For example, assume value 129 is returned for the node enable register. The binary equivalent is 10000001. This value indicates that bit B0 (MSB) and bit B7 (OSB) are set. • Assigning a value to this attribute enables one or more status events for service request. When an enabled status event occurs, bit B6 of the status byte sets to generate an SRQ (service request). • The service request enable register uses most of the same summary events as the status byte. Bit B6 (MSS) is not used by the enable register. For details, see "status.condition" register.
Details	See "Status byte and service request (SRQ)" in Appendix D.
Example	Sets the MSB bit of the service request enable register: `status.request_enable = status.MSB`

status.request_event	
Attribute	Service request event register.
Usage	Reads the service request event register: `serveventreg = status.request_event`
Remarks	• This attribute is used to read the service request event register, which is returned as a numeric value. Reading this register returns a value. The binary equivalent of the returned value. The least significant bit of the binary number is bit 0, and the most significant bit is bit 7. • For example, assume value 129 is returned for the event register. The binary equivalent is 10000001. This value indicates that bit B0 (MSB) and bit B7 (OSB) are set. • The service request event register uses most of the same summary events as the status byte. Bit B6 (MSS) is not used by the event register. For details, see "status.condition" register.

Details	See "Status byte and service request (SRQ)" in Appendix D.
Example	Reads the service request event register: serveventreg `= status.`request_event `print(`serveventreg`)` Output: `1.29000e+02` The above output indicates that bits B0 (MSS) and B7 (OSB) are set.

status.reset	
Function	Resets all bits set in the status model.
Usage	`status.reset()`
Remarks	This function clears all status data structure registers (enable, event, NTR and PTR) to their power up states.
Details	Appendix D.

status.standard.*
status.standard.condition
status.standard.enable
status.standard.event
*** = condition, enable or event**

Attribute	Standard event register set.

Usage	Reads condition, enable and event registers:
	`standardreg = status.standard.condition`
	`standardreg = status.standard.enable`
	`standardreg = status.standard.event`
	Writes to enable register:
	`status.standard.enable = standardreg`
	Set `standardreg` to one of the following values:
	`0` Clears all bits.
	`status.standard.OPERATION_COMPLETE` Sets OPC bit (B0).
	`status.standard.OPC` Sets OPC bit (B0).
	status.standard.QUERY_ERROR Sets QYE bit (B2).
	status.standard.QYE Sets QYE bit (B2).
	status.standard.DEVICE_DEPENDENT_ERROR Sets DDE bit (B3).
	status.standard.DDE Sets DDE bit (B3).
	status.standard.EXECUTION_ERROR Sets EXE bit (B4).
	status.standard.EXE Sets EXE bit (B4).
	status.standard.COMMAND_ERROR Sets CME bit (B5).
	status.standard.CME Sets CME bit (B5).
	status.standard.USER_REQUEST Sets URQ bit (B6).
	status.standard.URQ Sets URQ bit (B6).
	status.standard.POWER_ON Sets PON bit (B7).
	status.standard.PON Sets PON bit (B7).
	`standardreg` can also be set to the decimal weight of the bit to be set. Examples:
	To set bit B0 (OPC), set `standardreg` to 1 (2^0).
	To set bit B2 (QYE), set `standardreg` to 4 (2^2).
	To set bit B5 (CME), set `standardreg` to 32 (2^5).
	To set more than one bit of the register, set `standardreg` to the sum of their decimal weights. For example, to set bits B0 and B2, set `standardreg` to 5 (1 + 4).

Remarks	• This attribute is used to read or write to the standard event registers.
	• Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit 0, and the most significant bit is bit 15.
	• For example, assume value 9 is returned for the enable register. The binary equivalent is 0000000000001001. This value indicates that bit 0 (OPC) and bit 3 (DDE) are set.
	• The used bits of the standard event status register are described as follows:
	  • **Bit B0, Operation Complete (OPC) –** 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 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.

Details	See "Standard Event Register" in Appendix D.
Example	Sets the PON bit of the standard event enable register: `status.standard.enable = status.standard.PON`

status.system.* status.system.condition status.system.enable status.system.event * = condition, enable or event	
Attribute	TSP-Link system data structure register set.

Usage	Reads condition, enable and event registers:
	`enablereg = status.system.condition`
	`enablereg = status.system.enable`
	`enablereg = status.system.event`
	Writes to enable register:
	`status.system.enable = enablereg`
	Set `enablereg` to one of the following values:
	`0` Clears all bits.
	`1` or `status.system.EXTENSION_BIT` Sets EXT bit (B0).
	`1` or `status.system.EXT` Sets EXT bit (B0).
	`status.system.NODEn` Sets a node bit (Bn); `n` = 1 to 14.

Remarks	• In an expanded system (TSP-Link), this attribute is used to read or write to the system node registers. • Reading a system node register returns a numeric value whose binary equivalent indicates which register bits are set. The bits of the system node register are identified as follows: Bit B0 – EXT bitBit B4 – Node 4Bit B8 – Node 8Bit B12 – Node 12 Bit B1 – Node 1Bit B5 – Node 5Bit B9 – Node 9 Bit B13 – Node 13 Bit B2 – Node 2Bit B6 – Node 6Bit B10 – Node 10 Bit B14 – Node 14 Bit B3 – Node 3Bit B7 – Node 7Bit B11 – Node 11 Bit B15 – Not used • For example, assume value 9 is returned for the enable register. The binary equivalent is 0000000000001001. This value indicates that bit 0 (EXT) and bit 3 (Node 3) are set. • Assigning a value to the `status.system.enable` attribute sets the extension bit or a node bit of the system node enable register.
Details	See "System Summary Event Registers" in Appendix D.
Also see	status.system2.*, status.system3.*, status.system4.*, status.system5.*
Example	Sets the extension bit of the system enable register: `status.system.enable = status.system.EXT`

status.system2.*
status.system2.condition
status.system2.enable
status.system2.event
* = condition, enable or event

Attribute	TSP-Link system2 data structure register set.

Usage	Reads condition, enable and event registers:
	`enablereg = status.system2.condition`
	`enablereg = status.system2.enable`
	`enablereg = status.system2.event`
	Writes to enable register:
	`status.system2.enable = enablereg`
	Set `enablereg` to one of the following values:
	`0` Clears all bits.
	`1` or `status.system2.EXTENSION_BIT` Sets EXT bit (B0).
	`1` or `status.system2.EXT` Sets EXT bit (B0).
	`status.system2.NODEn` Sets a node bit (Bn); `n` = 15 to 28.

Remarks	• In an expanded system (TSP-Link), this attribute is used to read or write to the system2 node registers. • Reading a system2 node register returns a numeric value whose binary equivalent indicates which register bits are set. The bits of the system2 node register are identified as follows: Bit B0 – EXT bitBit B4 – Node 18Bit B8 – Node 22Bit B12 – Node 26 Bit B1 – Node 15Bit B5 – Node 19Bit B9 – Node 23Bit B13 – Node 27 Bit B2 – Node 16Bit B6 – Node 20Bit B10 – Node 24 Bit B14 – Node 28 Bit B3 – Node 17Bit B7 – Node 21Bit B11 – Node 25 Bit B15 – Not used • For example, assume value 9 is returned for the enable register. The binary equivalent is 0000000000001001. This value indicates that bit 0 (EXT) and bit 3 (Node 17) are set. • Assigning a value to the `status.system2.enable` attribute sets the extension bit or a node bit of the system2 node enable register.
Details	See "System Summary Event Registers" in Appendix D.
Also see	status.system.*, status.system3.*, status.system4.*, status.system5.*
Example	Sets the extension bit of the system2 enable register: `status.system2.enable = status.system2.EXT`

status.system3.* status.system3.condition status.system3.enable status.system3.event * = condition, enable or event	
Attribute	TSP-Link system3 data structure register set.

Usage	Reads condition, enable and event registers:
	`enablereg = status.system3.condition`
	`enablereg = status.system3.enable`
	`enablereg = status.system3.event`
	Writes to enable register:
	`status.system3.enable = enablereg`
	Set `enablereg` to one of the following values:
	`0` Clears all bits.
	`1` or `status.system3.EXTENSION_BIT` Sets EXT bit (B0).
	`1` or `status.system3.EXT` Sets EXT bit (B0).
	`status.system3.NODEn` Sets a node bit (Bn); `n` = 29 to 42.

Remarks	• In an expanded system (TSP-Link), this attribute is used to read or write to the system3 node registers. • Reading a system3 node register returns a numeric value whose binary equivalent indicates which register bits are set. The bits of the system3 node register are identified as follows: Bit B0 – EXT bitBit B4 – Node 32Bit B8 – Node 36Bit B12 – Node 40 Bit B1 – Node 29Bit B5 – Node 33Bit B9 – Node 37Bit B13 – Node 41 Bit B2 – Node 30Bit B6 – Node 34Bit B10 – Node 38 Bit B14 – Node 42 Bit B3 – Node 31Bit B7 – Node 35Bit B11 – Node 39 Bit B15 – Not used • For example, assume value 9 is returned for the enable register. The binary equivalent is 0000000000001001. This value indicates that bit 0 (EXT) and bit 3 (Node 31) are set. • Assigning a value to the `status.system3.enable` attribute sets the extension bit or a node bit of the system3 node enable register.
Details	See "System Summary Event Registers" in Appendix D.
Also see	status.system.*, status.system2.*, status.system4.*, status.system5.*
Example	Sets the extension bit of the system3 enable register: `status.system3.enable = status.system3.EXT`

status.system4.*
status.system4.condition
status.system4.enable
status.system4.event
* = condition, enable or event

Attribute	TSP-Link system4 data structure register set.

Usage	Reads condition, enable and event registers:
	`enablereg = status.system4.condition`
	`enablereg = status.system4.enable`
	`enablereg = status.system4.event`
	Writes to enable register:
	`status.system4.enable = enablereg`
	Set `enablereg` to one of the following values:
	`0` Clears all bits.
	`1` or `status.system4.EXTENSION_BIT` Sets EXT bit (B0).
	`1` or `status.system4.EXT` Sets EXT bit (B0).
	`status.system4.NODEn` Sets a node bit (Bn); `n` = 43 to 56.

Remarks	• In an expanded system (TSP-Link), this attribute is used to read or write to the system4 node registers. • Reading a system4 node register returns a numeric value whose binary equivalent indicates which register bits are set. The bits of the system4 node register are identified as follows: Bit B0 – EXT bitBit B4 – Node 46Bit B8 – Node 50Bit B12 – Node 54 Bit B1 – Node 43Bit B5 – Node 47Bit B9 – Node 51Bit B13 – Node 55 Bit B2 – Node 44Bit B6 – Node 48Bit B10 – Node 52 Bit B14 – Node 56 Bit B3 – Node 45Bit B7 – Node 49Bit B11 – Node 53 Bit B15 – Not used • For example, assume value 9 is returned for the enable register. The binary equivalent is 0000000000001001. This value indicates that bit 0 (EXT) and bit 3 (Node 45) are set. • Assigning a value to the `status.system4.enable` attribute sets the extension bit or a node bit of the system4 node enable register.
Details	See "System Summary Event Registers" in Appendix D.
Also see	status.system.*, status.system2.*, status.system3.*, status.system5.*
Example	Sets the extension bit of the system4 enable register: `status.system4.enable = status.system4.EXT`

status.system5.* status.system5.condition status.system5.enable status.system5.event * = condition, enable or event	
Attribute	TSP-Link system5 data structure register set.

Usage	Reads condition, enable and event registers:
	`enablereg = status.system5.condition`
	`enablereg = status.system5.enable`
	`enablereg = status.system5.event`
	Writes to enable register:
	`status.system5.enable = enablereg`
	Set `enablereg` to one of the following values:
	`0` Clears all bits.
	`1` or `status.system5.EXTENSION_BIT` Sets EXT bit (B0).
	`1` or `status.system5.EXT` Sets EXT bit (B0).
	`status.system5.NODEn` Sets a node bit (Bn); `n` = 57 to 64.

Remarks	• In an expanded system (TSP-Link), this attribute is used to read or write to the system5 node registers. • Reading a system5 node register returns a numeric value whose binary equivalent indicates which register bits are set. The bits of the system5 node register are identified as follows: Bit B0 – EXT bitBit B4 – Node 60Bit B8 – Node 64 Bit B1 – Node 57Bit B5 – Node 61Bits B9 through B15 – Not used Bit B2 – Node 58Bit B6 – Node 62 Bit B3 – Node 59Bit B7 – Node 63 • For example, assume value 9 is returned for the enable register. The binary equivalent is 0000000000001001. This value indicates that bit 0 (EXT) and bit 3 (Node 59) are set. • Assigning a value to the `status.system5.enable` attribute sets the extension bit or a node bit of the system5 node enable register.
Details	See "System Summary Event Registers" in Appendix D.
Also see	status.system.*, status.system2.*, status.system3.*, status.system4.*
Example	Sets the extension bit of the system5 enable register: `status.system5.enable = status.system5.EXT`

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.

timer.measure.t	
Function	Measures the elapsed time since the timer was last reset.
Usage	`time = timer.measure.t()` time `Returns the elapsed time in seconds. (1µs resolution).`
Remarks	• This function will return the elapsed time in seconds since the timer was reset. • The returned resolution for `time` depends on how long it has been since the timer was reset. It starts with 1µs resolution and starts to lose resolution after about 2.8 minutes.
Also see	timer.reset

Example	Resets the timer and then measures the time since the reset:
	`timer.reset()`
	`...`
	`time = timer.measure.t()`
	`print(time)`
	Output: `1.469077e+01`
	The above output indicates that `timer.measure.t` was executed 14.69077 seconds after `timer.reset`.

timer.reset	
Function	Resets the timer to 0 seconds.
Usage	`timer.reset()`
Remarks	This function will restart the timer at zero.
Also see	timer.measure.t

Example	Resets the timer and then measures the time since the reset:
	`timer.reset()`
	`...`
	`time = timer.measure.t()`
	`print(time)`
	Output: `1.469077e+01`
	The above output indicates that `timer.measure.t` was executed 14.69077 seconds after `timer.reset`.

trigger functions

The functions in this group are used to control triggering.

trigger.clear	
Function	Clears the command interface trigger event detector.
Usage	`trigger.clear()`
Remarks	The trigger event detector remembers if an event has been detected since the last `trigger.wait` call. This function clears the trigger's event detector and discards the previous history of command interface trigger events.
Details	See "Triggering" in Section 10.
Also see	trigger.wait

trigger.wait	
Function	Wait for a trigger event.
Usage	`triggered = trigger.wait(timeout)` timeout — Maximum amount of time in seconds to wait for the trigger. triggered — Returns `true` if a trigger was detected. Returns `false` if no triggers were detected during the timeout period.
Remarks	• This function will wait up to `timeout` 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. • 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.

Details	See "Triggering" in Section 10.
Also see	trigger.clear

Example	Waits up to 10 seconds for a trigger:
	`triggered = trigger.wait(10)`
	`print(triggered)`
	Output: `false`
	The above output indicates that no trigger was detected during the 10 second timeout.

tsplink function and attributes

Use the function and attributes in this section to assign node numbers to the Series 2600 instruments and initialize the TSP-Link system.

tsplink.group	
Attribute	Stores the group number assigned to the node.

Usage	`groupnumber = tsplink.group`
	`tsplink.group = groupnumber`
	
	groupnumber The TSP-Link group number.

Remarks	• The default group number is 0. • Set the attribute value to 0 to remove the node from all groups. • Each time the node powers off, the group number for that node changes to 0. • Group numbers can range from 0 to 64.

tsplink.master	
Attribute	Reads the node number assigned to the master node.
Usage	`master = tsplink.master` `master` A custom variable that stores the master node number.
Remarks	A read-only attribute.

tsplink.node	
Attribute	Use this attribute to set the node number. Choose a number from 1 to 64.
Usage	`mynode = tsplink.node` Reads the node number. `tsplink.node = mynode` Writes the node number. mynode Sets the node number. Choose from 1 to 64.
Remarks	• This attribute sets the TSP-Link node number and saves the value in nonvolatile memory. • Changes to the node number does not take effect until the next time `tsplink.reset` is executed on any node in the system. • Each node connected to the TSP-Link must be assigned a different node number. • 16 nodes is the maximum number of nodes allowed
Details	See Section 9 System expansion.
Also see	tsplink.reset, tsplink.state
Example	`--Sets the TSP-Link node to number 2:` `tsplink.node = 2`

tsplink.readbit	Replace N with the number of the synchronization line: 1 to 3.
Function	Reads the state of a TSP-Link synchronization line.
Usage	`data = tsplink.readbit(N)` data — A custom variable that stores the state of the synchronization line. N — Synchronization line to be read (1 - 3).
Remarks	Returns a value of 0 if the line is low and a 1 if the line is high.
Details	See "TSP-Link Synchronization lines" in Section 10.
Also see	tsplink.readbit, tsplink.readport, tsplink.writebit.

Example	`--Assume line 3 is set high, and it is then read:`
	`data = tsplink.readbit(3)`
	`print(data)`
	Output: `1.000000e+00`

tsplink.readport	
Function	Reads the TSP-Link synchronization lines as a digital I/O port.
Usage	`data = tsplink.readport()`
Remarks	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 3 corresponds to line 3. For example, a returned value of 2 has a binary equivalent of 010. Line 2 is high (1), and the other 2 lines are low (0).
Details	See "TSP-Link Synchronization lines" in Section 10.
Also see	tsplink.readbit, tsplink.writebit, and tsplink.writeport

Example	Assume line 2 is set high, and the digio I/O port reads:
	`data = tsplink.readport()`
	`print(data)`
	Output: `2.000000e+00` (binary 010)

tsplink.reset	
Function	Initializes (resets) all nodes (instruments) in the TSP-Link system.
Usage	`tsplink.reset()`
Remarks	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.
Details	See Section 9 “System expansion”.
Also see	tsplink.node, tsplink.state

tsplink.state	
Attribute	TSP-Link on-line state.
Usage	`state = tsplink.state`
Remarks	• This attribute stores the TSP-Link status, either `online` or `offline`. The state will be “offline” after the unit is powered on. After `tsplink.reset` is successful, the state will be “online”. • This attribute is read-only.
Details	See Section 9 “System expansion”.
Also see	tsplink.node, tsplink.reset

Example	Reads the on-line state of the TSP-Link:
	`state = tsplink.state`
	`print(state)`
	Output: `online`

tsplink.trigger[N].assert	Replace N with the number of the synchronization line: 1 to 3.
Function	Asserts a trigger on one of the synchronization lines.
Usage	`tsplink.trigger[N].assert()`
Remarks	The set pulse width determines how long the trigger is asserted.
Details	See "TSP-Link Synchronization lines" in Section 10.
Also see	tsplink.trigger[N].pulsewidth
Example	-- Asserts trigger on I/O line 2: `tsplink.trigger[2].assert()`

tsplink.trigger[N].clear	Replace N with the number of the synchronization line: 1 to 3.
Function	Clears a trigger event on a synchronization line.
Usage	`tsplink.trigger[N].clear()`
Remarks	• The Trigger event detection recalls if a trigger event has been detected since the last `tsplink.trigger[N].wait` call. • This function clears a trigger event detector, discards the previous history of the trigger line, and clears the `tsplink.trigger[N].overrun` attribute.
Details	See "TSP-Link Synchronization lines" in Section 10.
Also see	`tsplink.trigger[N].wait`
Example	-- Clears trigger event on synchronization line 2: `tsplink.trigger[2].clear()`

tsplink.trigger[N].mode	Replace N with the number of the synchronization line: 1-3
Function	`The trigger operation and detection mode.`

Usage	`mode = tsplink.trigger[N].mode`
	`tsplink.trigger[N].mode = mode`
	
	N — The number of the trigger line.
	trig_mode — The active trigger mode.
	mode — Selects the current trigger mode.
	
	Choose one the following values for mode:
	`0` or `tsplink.TRIG_BYPASS` — Allows direct control of the line.
	`1` or `tsplink.TRIG_FALLING` — Detects falling edge input triggers. Asserts TTL-low pulse as an output trigger.
	`2` or `tsplink.TRIG_RISING` — If the programmed state of the line is high, the `tsplink.TRIG_RISING` mode behaves similar to `tsplink.TRIG_RISINGA`. If the programmed state of the line is low, the `tsplink.TRIG_RISING` mode behaves similar to `tsplink.TRIG_RISINGM`.
	`3` or `tsplink.TRIG_EITHER` — Detects rising or falling edge triggers. Asserts a TTL-low trigger pulse.
	`4` or `tsplink.TRIG_SYNCHRONOUSA` — Detects the falling edge input triggers and automatically latches and drives the trigger line low. Asserting the output trigger releases the latched line.
	`5` or `tsplink.TRIG_SYNCHRONOUS` — Detects the falling edge input triggers and automatically latches and drives the trigger line low. Asserts a TTL-low pulse as an output trigger.
	`6` or `tsplink.TRIG_SYNCHRONOUSM` — Detects rising edge triggers as an input. Asserts a low TTL-low pulse for output.
	`7` or `tsplink.TRIG_RISINGA` — Detects Rising Edge triggers as an input. Asserts a low TTL-low pulse as an output.
	`8` or `tsplink.TRIG_RISINGM` — Edge detection as an input is not available. Generates a TTL-high pulse as an output trigger.

Remarks	• You can express the mode as a number (0 through 8) or you can use one of the pre-defined constants (see "Usage"). • The custom variable mode stores the trigger mode as a numeric value when the attribute is read. • The default trigger mode for a line is `tsplink.TRIG_BYPASS`. • To control the line state, use the `tsplink.TRIG_BYPASS` mode with the `tsplink.writebit` and the `tsplink.writeport` commands. • (Firmware version prior to 1.4.0 only) Use the `tsplink.TRIG_SYNCHRONOUS` mode for backward firmware compatibility. • (Firmware version 1.4.0 and higher) Use `tsplink.TRIG_SYNCHRONOUSA` or `tsplink.TRIG_SYNCHRONOUSM` modes. • (Firmware version prior to 1.4.0 only) Use the `tsplink.TRIG_RISING` mode for backward compatibility. • (Firmware version 1.4.0 and higher) Use `tsplink.TRIG_RISINGA` or `tsplink.TRIG_RISINGM` modes.
Details	See"TSP-Link Synchronization lines" in Section 10, digio.writebit, and digio.writeport.
Example	-- Sets the trigger mode for the synchronization line 3: `tsplink.TRIG_RISINGM` `tsplink.trigger [3] = 8`

tsplink.trigger[N].overrun	Replace N with the number of the synchronization line: 1-3
Attribute	Indicates if the event detector ignored an event while in the detected state.
Usage	`overrun = tsplink.trigger[N].overrun` N — The trigger line. overrun — The trigger overrun state.
Remarks	• A read-only attribute. • Indicates an event was ignored because the event detector was in the detect state when the event was detected.

tsplink.trigger[N].pulsewidth	Replace N with the number of the synchronization line: 1 to 3.
Attribute	The length of time that the trigger line will be asserted for output triggers.
Usage	`width = tsplink.trigger[N].pulsewidth` Reads pulse width. `tsplink.trigger[N].pulsewidth = width` Writes pulse width. `width` — The pulse width length (seconds).
Remarks	• Setting the pulse width to 0 (seconds) asserts the trigger indefinitely. • The default pulsewidth time is 10µs.
Details	See "TSP-Link Synchronization lines" in Section 10.
Also see	tsplink.trigger[N].release
Example	-- Sets pulse width for trigger line 3 to 20µs: `tsplink.trigger[3].pulsewidth = 20e-6`

tsplink.trigger[N].release	Replace N with the number of the synchronization line: 1 to 3.
Function	Releases an indefinite length or latched trigger.
Usage	`tsplink.trigger[N].release()` N — The trigger line number.
Remarks	Releases a trigger that was asserted with an indefinite pulse width, as well as a trigger that was latched in response to receiving a synchronous mode trigger.
Details	See "TSP-Link Synchronization lines" in Section 10.
Also see	tsplink.trigger[N].pulsewidth
Example	-- Releases trigger line 3: `tsplink.trigger[3].release()`

tsplink.trigger[N].wait	Replace N with the number of the synchronization line: 1 to 3.
Function	Waits for a trigger.
Usage	`triggered = tsplink.trigger[N].wait(timeout)` timeout — Specifies the time-out value in seconds. triggered — A customized variable that stores the value true if a trigger is detected, or false if a trigger is not detected during the time-out period.
Remarks	This function waits up to `timeout` value in seconds for an input trigger. If one or more trigger events were detected since the last time `tsplink.trigger[N].wait` or `tsplink.trigger[N].clear` was called, this function will return immediately. After waiting for a trigger with this function, the event detector is automatically rearmed and reset. This functionality is true regardless of the number of events detected.
Details	See "TSP-Link Synchronization lines" in Section 10.
Also see	tsplink.trigger[N].clear

tsplink.writebit	
Function	Sets a TSP-Link synchronization line high or low.
Usage	tsplink.writebit(bit, data) N — The synchronization line number (1 to 3). data — Value to write to the bit; 0 (low) or 1 (high).
Remarks	• If the output line is write protected by the `tsplink.writeprotect` attribute, the command will be ignored. • The `reset` function does not affect the present states of the digital I/O lines. • Use the `tsplink.writebit` and `tsplink.writeport` commands to control the output state of the synchronization line when the trigger mode is set to `tsplink.TRIG_BYPASS`.
Details	See "TSP-Link Synchronization lines" in Section 10.
Also see	tsplink.readbit, tsplink.readport, tsplink.writebit.
Example	Sets synchronization line 3 low (0): `tsplink.writebit(3, 0)`

tsplink.writeport	
Function	Writes to all TSP-Link synchronization lines.
Usage	`tsplink.writeport(data)` data — Value to write to the port; 0 to 7.
Remarks	• The binary representation of data indicates the output pattern to be written to the I/O port. For example, a `data` value of 2 has a binary equivalent of 010. Line 2 is set high (1), and the other 2 lines are set low (0). • Write protected lines will not be changed (see `tsplink.writeprotect`). • The `reset` function does not affect the present states of the digital I/O lines. • Use the `tsplink.writebit` and `tsplink.writeport` commands to control the output state of the synchronization line while the trigger mode is set to `tsplink.TRIG_BYPASS`.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	tsplink.readbit, tsplink.readport, tsplink.writebit.
Example	Sets the synchronization lines 1 and 2 high (binary 011): `tsplink.writeport(3)`

tsplink.writeprotect	
Attribute	Write protect mask that disables bits from being changed with the `tsplink.writebit` and `tsplink.writeport` functions.
Usage	`mask = tsplink.writeprotect` -- Reads write protect mask. `tsplink.writeprotect = mask` -- Writes write protect mask. `mask` --Set to the value that specifies the bit pattern for write protect.
Remarks	• Bits set to 1 cause the corresponding line to be write protected. • The binary equivalent of `mask` indicates the mask to be set for the I/O port. For example, a `mask` value of 5 has a binary equivalent 101. This mask write protects lines 1 and 3.
Details	See "Controlling digital I/O lines" in Section 10.
Also see	tsplink.readbit, tsplink.readport tsplink.writeport
Example	Write protects lines 1 and 3: `tsplink.writeprotect = 5`

userstring functions

The functions in this group are used to store/retrieve user-defined strings in non-volatile memory.

userstring.add	
Function	Adds a user-defined string to non-volatile memory.
Usage	`userstring.add(name, value)` name — The name for the string. value — The string to associate with the name.
Remarks	This function will associate the string `value` with the string `name` and store the pair in non-volatile memory. The value associated with the given name can be retrieved with the `userstring.get` function.
Also see	userstring.catalog, userstring.delete, userstring.get

Example	Stores user-defined strings in non-volatile memory:
	`userstring.add("assetnumber", "236")`
	`userstring.add("department", "Widgets")`
	`userstring.add("contact", "John Doe")`

userstring.catalog	
Function	Creates an iterator for the user string catalog.
Usage	`for name in userstring.catalog() do ... end`
Remarks	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.
Also see	userstring.add, userstring.delete, userstring.get

Example	Deletes all user strings in non-volatile memory:
	`for name in userstring.catalog() do`
	`  userstring.delete(name)`
	`end`
	Prints all user string name value pairs in non-volatile memory:
	for name in userstring.catalog() do
	`  print(name .. " = " .. userstring.get(name))`
	`end`
	Output: `department = Widgets`
	`  assetnumber = 236`
	`  contact = John Doe`
	The above output lists the user strings added in the “Example” for the userstring.add function. Notice that they are not listed in the order that they were added.

userstring.delete	
Function	Deletes a user-defined string from non-volatile memory.
Usage	`userstring.delete(name)` name Name of the user string.
Remarks	This function will delete from non-volatile memory the string that is associated with the string `name`.
Also see	userstring.add, userstring.catalog, userstring.get

Example	Deletes user-defined strings from non-volatile memory:
	`userstring.delete("assetnumber")`
	`userstring.delete("department")`
	`userstring.delete("contact")`

userstring.get	
Function	Retrieves a user-defined string from non-volatile memory.
Usage	`value = userstring.get(name)` `name` Name of the user string. value Returns the string value associated with name.
Remarks	This function will retrieve from non-volatile memory the string that is associated with the string name.
Also see	userstring.add, userstring.catalog, userstring.delete

Example	Retrieves the value for a user string from non-volatile memory:
	`value = userstring.get("assetnumber")`
	`print(value)`
	Output: `236`

waitcomplete function

This function waits for all overlapped commands to complete.

waitcomplete	
Function	Waits for all overlapped commands to complete.
Usage	`waitcomplete([group])` group Identifies the number of the group to wait for overlapped operations to complete.
Remarks	• The master node is the only node that can specify a group number with the `waitcomplete` command. • If a TSP-Link group number is not provided, the local group number is used. • If a 0 (zero) is used for the TSP-Link group number, this function waits for all nodes on the TSP-Link network. • Use this function to wait for a specific group to complete all overlapped operations. ***NOTE*** *Any node that is not assigned to a group (indicates the group number is 0) is considered part of the master group.* *Using this function without a group number waits for overlapped operations to complete on the local group.*
Example	waitcomplete() Waits for all nodes in the local group. waitcomplete(G) Waits for all nodes in group G. waitcomplete(0) Waits for all nodes on the TSP-Link network.

Section 13

Factory Scripts

In this section:

Topic	Page
Introduction	**13-2**
Factory script	**13-2**
KIGeneral	13-2
KIPulse	13-12
Advanced features for Models 2635 and 2636	**13-13**
Flash firmware upgrade	**13-35**

Introduction

The Series 2600 is shipped with one or more Factory Scripts saved in its flash firmware memory. A factory script is made up of a number of functions. Some of them can be called from the front panel LOAD TEST menu. All of them can be called using remote programming.

As Keithley Instruments develops additional factory scripts, they will be made available on the Keithley Instruments web site (www.keithley.com) as a flash firmware upgrade for the Series 2600. See "Flash firmware upgrade" for instructions on upgrading the flash firmware of your Series 2600.

Factory script

KIGeneral

The KIGeneral factory script is made up of the following functions. All of these functions can be accessed from both the front panel and the remote interfaces.

```
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.

Table 13-1
KIGeneral TSP test script: PulseIMeasureV

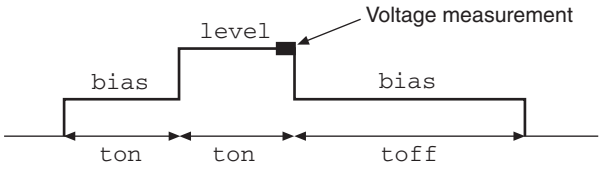
TSP project name: TSP test script name: Firmware version:	KIFactoryGeneral KIGeneral 1.0.2 and later
Function: `PulseIMeasureV(smu, bias, level, ton, toff, points)`	
Description	Performs a specified number of pulse I, measure V cycles: • Sets the `smu` to output `bias` amps and dwell for `ton` seconds. • Sets the `smu` to output `level` amps and dwell for `ton` seconds. • Performs voltage measurement with source at `level` amps. • Sets the `smu` to output `bias` amps for `toff` seconds. • Repeats the above sequence for `points` pulse-measure cycles. 

Table 13-1 (continued)
KIGeneral TSP test script: PulseIMeasureV

TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Function: `PulseIMeasureV(smu, bias, level, ton, toff, points)`	
Parameters	`smu, bias, level, ton, toff, points` `smu`: SourceMeter Channel (A or B). Defaults to SMU A 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.

Table 13-2
KIGeneral TSP test script: PulseVMeasureI

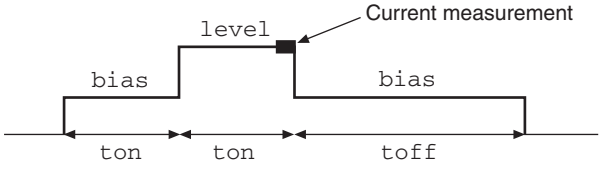
TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Function: `PulseVMeasureI(smu, bias, level, ton, toff, points)`	
Description	Performs a specified number of pulse V, measure I cycles: • Sets the `smu` to output `bias` volts and dwell for `ton` seconds. • Sets the `smu` to output `level` volts and dwell for `ton` seconds. • Performs current measurement with source at `level` amps. • Sets the `smu` to output `bias` volts for `toff` seconds. • Repeats the above sequence for `points` pulse-measure cycles. 

Table 13-2 (continued)
KIGeneral TSP test script: PulseVMeasureI

TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Parameters	`smu, bias, level, ton, toff, points` `smu`: SourceMeter Channel (A or B). Defaults to SMU A if all parameters are omitted when function is called. `bias`: Bias level in volts. `level`: Pulse level in volts. `ton`: Pulse on-time in seconds. `toff`: Pulse off-time in seconds. `points`: Number of pulse-measure cycles.
Data	Pulsed current measurements, voltage levels and timestamps are stored in `smuX.nvbuffer1`.
Example	`PulseVMeasureI(smub, -1, 1, 1E-3, 2E-3, 20)` 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.

Table 13-3
KIGeneral TSP test script: SweepILinMeasureV

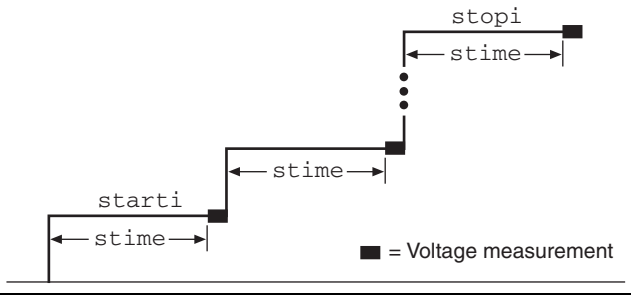
TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Function: `SweepILinMeasureV(smu, starti, stopi, stime, points)`	
Description	Performs a linear current sweep with voltage measured at every step (point): • Sets the `smu` to output `starti` amps, allows the source to settle for `stime` seconds, and then performs a voltage measurement. • Sets the `smu` to output the next amps step, allows the source to settle for `stime` seconds, and then performs a voltage measurement. • Repeats the above sequence until the voltage is measured on the `stopi` amps step. The linear step size is automatically calculated as follows: *step = (stopi – starti) / (points – 1)* 

Table 13-3 (continued)
KIGeneral TSP test script: SweepILinMeasureV

TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Parameters	`smu, starti, stopi, stime, points` `smu`: SourceMeter Channel (A or B). Defaults to SMU A if all parameters are omitted when function is called. `starti`: Sweep start current in amps. `stopi`: Sweep stop current in amps. `stime`: Settling time in seconds. Occurs after stepping the source and before performing a measurement. `points`: Number of sweep points (must be ≥2).
Data	Voltage measurements, current source values and timestamps are stored in `smuX.nvbuffer1`.
Example	`SweepILinMeasureV(smua, -1E-3, 1E-3, 0, 100)` 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 `stime` is set for 0s, voltage will be measured as fast as possible after each current step.

Table 13-4
KIGeneral TSP test script: SweepVLinMeasureI

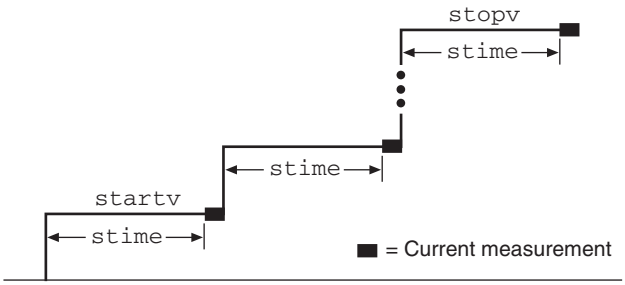
TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Function: `SweepVLinMeasureI(smu, startv, stopv, stime, points)`	
Description	Performs a linear voltage sweep with current measured at every step (point): • Sets the `smu` to output `startv` volts, allows the source to settle for `stime` seconds and then performs a current measurement. • Sets the `smu` to output the next volts step, allows the source to settle for `stime` seconds and then performs a current measurement. • Repeats the above sequence until the current is measured on the `stopv` volts step. The linear step size is automatically calculated as follows: *step = (stopv – startv) / (points – 1)* 

Table 13-4 (continued)
KIGeneral TSP test script: SweepVLinMeasureI

TSP project name:	KIFactoryGeneral
TSP test script name:	KIGeneral
Firmware version:	1.0.2 and later
Parameters	`smu, startv, stopv, stime, points` `smu:` SourceMeter Channel (A or B). Defaults to SMU A if all parameters are omitted when function is called. `starti:` Sweep start voltage in volts. `stopi:` Sweep stop voltage in volts. `stime:` Settling time in seconds. Occurs after stepping the source and before performing a measurement. `points:` Number of sweep points (must be ≥2).
Data	Current measurements, voltage source values and timestamps are stored in `smuX.nvbuffer1`.
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.

Table 13-5
KIGeneral TSP test script: SweepILogMeasureV

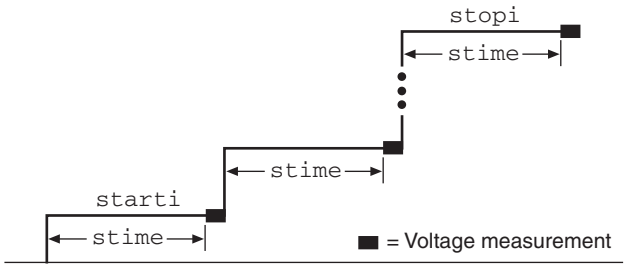
TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Function: `SweepILogMeasureV(smu, starti, stopi, stime, points)`	
Description	Performs a logarithmic current sweep with voltage measured at every step (point): • Sets the `smu` to output `starti` amps, allows the source to settle for `stime` seconds and then performs a voltage measurement. • Sets the `smu` to output the next amps step, allows the source to settle for `stime` seconds and then performs a voltage measurement. • Repeats the above sequence until the voltage is measured on the `stopi` amps step. The source level at each step (SourceStepLevel) is automatically calculated as follows: **MeasurePoint** = The step point number for a measurement. For example, for a 5-point sweep (`points` = 5), a measurement will be performed at MeasurePoint 1, 2, 3, 4 and 5. ***LogStepSize*** = *(log10(stopi) – log10(starti)) / (points – 1)* ***LogStep*** = *(MeasurePoint – 1) x (LogStepSize)* ***SourceStepLevel*** = *antilog(LogStep) x starti* 
Parameters	`smu, starti, stopi, stime, points` `smu:` SourceMeter channel (A or B). Defaults to SMU A if all parameters are omitted when function is called. `starti:` Sweep start current in amps. `stopi:` Sweep stop current in amps. `stime:` Settling time in seconds. Occurs after stepping the source and before performing a measurement. `points:` Number of sweep measure points (must be ≥2).
Data	Voltage measurements, current source values and timestamps are stored in `smuX.nvbuffer1`.

Table 13-5 (continued)
KIGeneral TSP test script: SweepILogMeasureV

TSP project name:	KIFactoryGeneral
TSP test script name:	KIGeneral
Firmware version:	1.0.2 and later
Function: `SweepILogMeasureV(smu, starti, stopi, stime, points)`	
Example	`SweepILogMeasureV(smua, 0.01, 0.1, 0.001, 5)` 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. The following log values and corresponding source levels for the 5-point log sweep are listed as follows:

MeasurePoint	LogStepSize	LogStep	SourceStepLevel
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

Table 13-6
KIGeneral TSP test script: SweepVLogMeasureI

TSP project name:	KIFactoryGeneral
TSP test script name:	KIGeneral
Firmware version:	1.0.2 and later
Function: `SweepVLogMeasureI(smu, startv, stopv, stime, points)`	

Table 13-6 (continued)
KIGeneral TSP test script: SweepVLogMeasureI

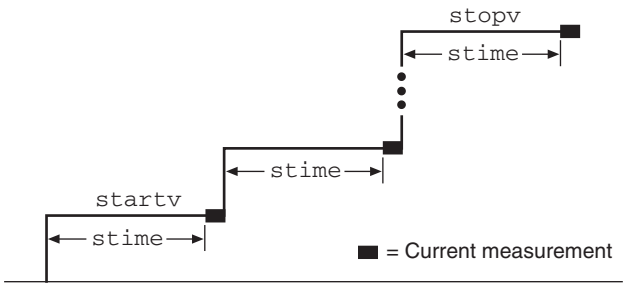
TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Description	Performs a logarithmic voltage sweep with current measured at every step (point): • Sets the `smu` to output `startv` volts, allows the source to settle for `stime` seconds and then performs a current measurement. • Sets the `smu` to output the next volts step, allows the source to settle for `stime` seconds and then performs a current measurement. • Repeats the above sequence until the current is measured on the `stopi` amps step. The source level at each step (SourceStepLevel) is automatically calculated as follows: ***MeasurePoint*** *= The step point number for a measurement. For example, for a 5-point sweep (points = 5), a measurement will be performed at MeasurePoint 1, 2, 3, 4 and 5.* ***LogStepSize*** *= (log10(stopi) – log10(starti)) / (points – 1)* ***LogStep*** *= (MeasurePoint – 1) x (LogStepSize)* ***SourceStepLevel*** *= antilog(LogStep) x starti* 
Parameters	`smu, startv, stopv, stime, points` `smu:` SourceMeter channel (A or B). Defaults to SMU A if all parameters are omitted when function is called. `starti:` Sweep start voltage in amps. `stopi:` Sweep stop voltage in amps. `stime:` Settling time in seconds. Occurs after stepping the source and before performing a measurement. `points:` Number of sweep measure points (must be ≥2).
Data	Current measurements, voltage source values and timestamps are stored in `smuX.nvbuffer1`.

Table 13-6 (continued)
KIGeneral TSP test script: SweepVLogMeasureI

TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
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:

MeasurePoint	LogStepSize	LogStep	SourceStepLevel
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

Table 13-7
KIGeneral TSP test script: SweepIListMeasureV

TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Function: `SweepIListMeasureV(smu, ilist, stime, points)`	
Description	Performs a current list sweep with voltage measured at every step (point): • Sets the `smu` to output the first `ilist` amps value, allows the source to settle for `stime` seconds and then performs a voltage measurement. • Sets the `smu` to output the next `ilist` amps value, allows the source to settle for `stime` seconds and then performs a voltage measurement. • Repeats the above sequence until the voltage is measured for the last amps value. The last point in the list to be measured is `points`.

Parameters	`smu, ilist, stime, points`
	`smu:` SourceMeter Channel (A or B). Defaults to SMU A if all parameters are omitted when function is called.
	`ilist:` Arbitrary list of current source values:
	`ilist = {value1, value2, ....valueN}`
	`stime:` Settling time in seconds. Occurs after sourcing a value and before performing a measurement.
	`points:` Number of sweep measure points (must be ≥2).

Data	Voltage measurements, current source values and timestamps are stored in `smuX.nvbuffer1`.

Example	`myilist = {-100E-9, 100E-9, -1E-6, 1E-6, -1E-3, 1E-3}`
	`SweepIListMeasureV(smua, myilist, 500E-6, 6)`
	This function performs a 6-point current list sweep starting at the first point in `myilist`. Voltage is measured at every step (point) in the sweep. The source will be allowed to settle on each value for 500µs before a measurement is performed.

Table 13-8
KIGeneral TSP test script: SweepVListMeasureI

TSP project name: KIFactoryGeneral TSP test script name: KIGeneral Firmware version: 1.0.2 and later	
Function: `SweepVListMeasureI(smu, vlist, stime, points)`	
Description	Performs a voltage list sweep with current measured at every step (point): • Sets the `smu` to output the first `vlist` volts value, allows the source to settle for `stime` seconds and then performs a current measurement. • Sets the `smu` to output the next `vlist` volts value, allows the source to settle for `stime` seconds and then performs a current measurement. • Repeats the above sequence until the current is measured for the last volts value. The last point in the list to be measured is `points`.

Parameters	`smu, vlist, stime, points`
	`smu:` SourceMeter Channel (A or B). Defaults to SMU A if all parameters are omitted when function is called.
	`vlist:` Arbitrary list of voltage source values:
	`vlist = {value1, value2, ....valueN}`
	`stime:` Settling time in seconds. Occurs after sourcing a value and before performing a measurement.
	`points:` Number of sweep measure points (must be ≥2).

Data	Current measurements, voltage source values and timestamps are stored in `smuX.nvbuffer1`.

Example:	`myvlist = {-0.1, 0.1, -1, 1, -6, 6, -40, 40, 0, 0}`
	`SweepVListMeasureI(smua, myvlist, 500E-6, 10)`
	This function performs a 10-point voltage list sweep starting at the first point in `myvlist`. Current is measured at every step (point) in the sweep. The source will be allowed to settle on each value for 500µs before a measurement is performed.

KIPulse

The KIPulse factory script is made up of the functions listed below. These functions can only be accessed from the remote interface and cannot be accessed from the front panel. The primary purpose of this factory script is to provide high-speed, high-power pulse functions. Pulses up to 10A and 200W (not available in the Models 2635/2636) can be generated using the functions in this library. For optimal pulse performance, the front display should be in the user screen mode. This will prevent display updates from interfering with pulse testing. Do this by sending the following command:

```
display.screen = display.USER
```

NOTE This factory script only operates on the SourceMeter channels present in the SourceMeter executing the pulse functions. These functions will not operate correctly if the user attempts to access SourceMeter channels on the TSP Link.

```
ConfigPulseIMeasureV
ConfigPulseVMeasureI
ConfigPulseIMeasureVSweepLin
ConfigPulseVMeasureISweepLin
ConfigPulseIMeasureVSweepLog
ConfigPulseVMeasureISweepLog
QueryPulseConfig
InitiatePulseTest
InitiatePulseTestDual
```

The “Config” functions are used to configure a pulse train and assign the configuration to the `tag` parameter. The “Initiate” functions are used to execute the pulse train(s) assigned to its `tag` argument(s). The conditions listed in the table below must be true for these functions to execute successfully.

Table 13-9
Required true conditions for function execution

Config	InitiatePulseTest	InitiatePulseTestDual
Source autorange (I and V) off.	Output on.	Output on.
Measure autorange (I and V) off.	Enough free space in buffer.	Enough free space in buffer.
Measure NPLC < ton.	Buffer `appendmode` on when pulse train is >1 point.	Buffer `appendmode` on when pulse train is >1 point.
Measure autozero OFF or ONCE.	Safety interlock engaged when using the 200V range.	Safety interlock engaged when using the 200V range.
		Different unique SMUs for each `tag`.
		Same NPLC setting for each `tag`.
		Same `toff` for each `tag`.
		`tag1 ton` ≥ `tag2 ton` by at least 41µsec.

Advanced features for Models 2635 and 2636

NOTE These advanced features are available with Models 2635 and 2636 firmware version 1.3.0 and higher.

Variable off time between pulses in a pulse train

The KIPulse scripts will now accept the off_time parameter as a table as well as just a number. The table allows defining different off times to be used after each pulse.

The following should be noted:

1. If off_time is passed as a number or only a single value is used in the table, it will be used for all points in a multiple point pulse.
2. The number of times specified in the table must match the number of points called for in the sweep.
3. The times used in tables must match for dual channel pulsing.
4. Each specified off_time must adhere to the duty cycle limits for the specified pulsing region. Example:

```
local timelist1 = { 1, 2, 3, 4, 5 }
local timelist2 = { }
for i = 1,5 do timelist2[i] = math.pow(10, i) end
```

   – configure a pulse with 1 second on time and variable off time, no measurement

```
f,msg = ConfigPulseVMeasureI(smua, 0, 1, 100e-3, 1,
timelist1, 5, nil, 1)
```

Simultaneous IV measurement during pulse

The KIPulse ConfigPulseXMeasureY() functions will now optionally accept an extra reading buffer in order to activate simultaneous IV measurements during pulsing.

Previous usage of passing in a reading buffer or a nil (for no measurement) is still supported. Example:

```
rbi = smua.makebuffer(10)
rbv = smua.makebuffer(10)
rbi.appendmode = 1
rbv.appendmode = 1

rbs = { i = rbi, v = rbv }

f,msg = ConfigPulseVMeasureI(smua, 0, 10, 1e-3, 1e-3, 1e-3,
2, rbs, 1)
```

Additional hardware triggering parameters

New parameters added to the “Configure” functions after the sync_out parameter:

- *sync_in_timeout - specifies the length of time (in seconds) to wait for input trigger (defaults to 10s)
- *sync_in_abort - (true or false), whether to abort pulse if in put trigger is not received before timeout expires (defaults to true)
- If pulse aborts due to missed trigger, message returned to the user indicates timer timeout.

NOTE *These parameters are new to firmware version 1.3.0.

Table 13-10
KIGeneral TSP test script: ConfigPulseIMeasureV

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`ConfigPulseIMeasureV`
Usage:
`f, msg = ConfigPulseIMeasureV (smu,bias,level,limit,ton,toff,points,buffer,tag[,sync_in][,sync_out] [,sync_in_timeout][,sync_in_abort])`	

Description	Configures a current pulse train with a voltage measurement at each point. Measurement(s) will be made at the end of the `ton` time. This function does not cause the specified `smu` to output a pulse train. It simply checks to see if all of the pulse dimensions are achievable and if so, assigns the indicated `tag` or index to the pulse train. The `InitPulseTest(tag)` and `InitPulseTestDual(tag)` functions are used to initiate a pulse train assigned to a valid `tag`.
Parameters	`smu,bias,level,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out]` `smu:` SourceMeter channel. (e.g. `smua`). `bias:` Pulse bias level in amps. `level:` Pulse level in amps. `limit:` Voltage limit (i.e. compliance) in volts. `ton:` Pulse width (i.e. on-time) in seconds. `toff:` Pulse off time in seconds. `points:` Number of pulse-measure cycles. `buffer:` Reading buffer where pulsed measurements will be stored. If this is `nil` when the function is called, no measurements will be made when the pulse train is initiated. `tag:` Numeric identifier to be assigned to the defined pulse train. `sync_in:` Defines a digital I/O trigger input line. If programmed, the pulse train will wait for a trigger input before executing each pulse. This parameter is optional. `sync_out:` Defines a digital I/O trigger output line. If programmed, the pulse train will generate a trigger output immediately prior to the start of `ton`. This parameter is optional. `sync_in_timeout:` Specifies the length of time (in seconds) to wait for input trigger (defaults to 10s). *New to Firmware 1.3.0* `sync_in_abort:` (true or false) Whether to abort pulse if input trigger is not received before timeout expires (defaults to true). *New to Firmware 1.3.0*

Table 13-10 (continued)
KIGeneral TSP test script: ConfigPulseIMeasureV

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`ConfigPulseIMeasureV`
Usage:
`f, msg = ConfigPulseIMeasureV (smu,bias,level,limit,ton,toff,points,buffer,tag[,sync_in][,sync_out] [,sync_in_timeout][,sync_in_abort])`	

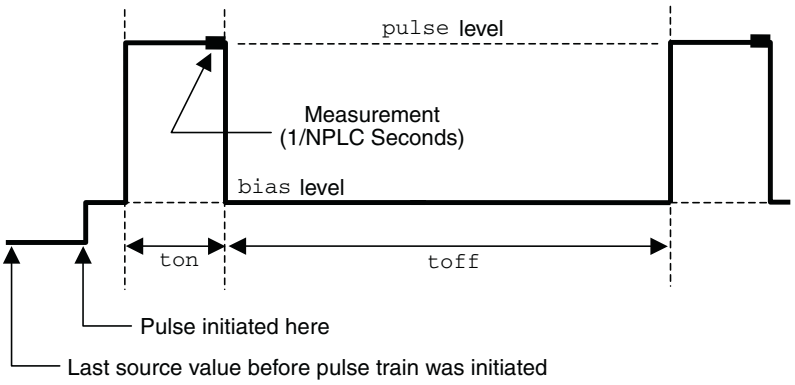
Waveform	
Return Values	`f`: A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered. `msg`: A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	Pulsed voltage measurements are stored in the reading buffer specified by the `buffer` input argument.
Example	`ConfigPulseIMeasureV(smua, 0, 5, 10, 0.001, 0.080, 1, smua.nvbuffer1, 1)` Set up a pulse train that will use SourceMeter channel A. The pulse amplitude will be 5A and will return to 0A after 1msec. The pulse will remain at 0A for 80 msec and the voltage limit will be 10V during the pulse. The pulse train will consist of only 1 pulse and this pulse will be assigned a tag index of 1.
See Also	KIGeneral TSP test script: InitiatePulseTest, `InitiatePulseTestDual`

Table 13-11
KIGeneral TSP test script: ConfigPulseVMeasureI

TSP project name:	KIFactoryPulse
TSP test script name:	KIPulse
Firmware version:	1.2.0 and later

Function:
`ConfigPulseVMeasureI`
Usage:
`f, msg = ConfigPulseVMeasureI (smu,bias,level,limit,ton,toff,points,buffer,tag[,sync_in][,sync_out] [,sync_in_timeout][,sync_in_abort])`	

Description	Configures a voltage pulse train with a current measurement at each point. Measurement(s) will be made at the end of the `ton` time. This function does not cause the specified `smu` to output a pulse train. It simply checks to see if all of the pulse dimensions are achievable and if so, assigns the indicated `tag` or index to the pulse train. `The InitPulseTest(tag)` and `InitPulseTestDual(tag)` functions are used to initiate a pulse train assigned to a valid tag.
Parameters	`smu,bias,level,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out]`
	`smu`: SourceMeter channel. (e.g. `smua`).
	`bias`: Pulse bias level in volts.
	`level`: Pulse level in volts.
	`limit`: Current limit (i.e. compliance) in amps.
	`ton`: Pulse width (i.e. on-time) in seconds.
	`toff`: Pulse off time in seconds.
	`points`: Number of pulse measure cycles.
	`buffer`: Reading buffer where pulsed measurements will be stored. If this is `nil` when the function is called, no measurements will be made when the pulse train is initiated.
	`tag`: Numeric identifier to be assigned to the defined pulse train.
	`sync_in`: Defines a digital I/O trigger input line. If programmed, the pulse train will wait for a trigger input before executing each pulse. This parameter is optional.
	`sync_out`: Defines a digital I/O trigger output line. If programmed, the pulse train will generate a trigger output immediately prior to the start of `ton`. This parameter is optional.
	`sync_in_timeout`: Specifies the length of time (in seconds) to wait for input trigger (defaults to 10s). *New to Firmware 1.3.0*
	`sync_in_abort`: (true or false) Whether to abort pulse if input trigger is not received before timeout expires (defaults to true). *New to Firmware 1.3.0*

Table 13-11 (continued)
KIGeneral TSP test script: ConfigPulseVMeasureI

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`ConfigPulseVMeasureI`
Usage:
`f, msg = ConfigPulseVMeasureI (smu,bias,level,limit,ton,toff,points,buffer,tag[,sync_in][,sync_out] [,sync_in_timeout][,sync_in_abort])`	

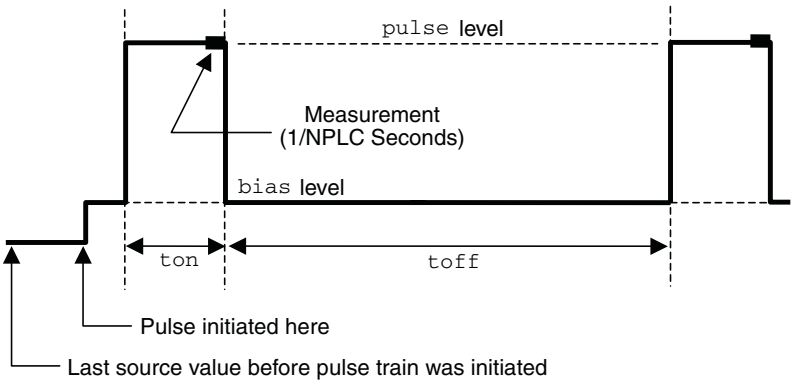
Waveform	
Return Values	`f`: A Boolean flag. This flag will be `true` when the pulse is successfully configured, `false` when errors were encountered. `msg`: A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	Pulsed current measurements are stored in the reading buffer specified by the `buffer` input argument.
Example	`ConfigPulseVMeasureI(smub, 0, 20, 1, 0.001, 0.080, 10, smub.nvbuffer1, 2)` Set up a pulse train that will use SourceMeter channel B. The pulse amplitude will be 20V and will return to 0V after 1msec. The pulse will remain at 0V for 80msec and the current limit will be 1A during the pulse. The pulse train will consist of 10 pulses and the pulse train will be assigned a tag index of 2.
See Also	`InitiatePulseTest`, `InitiatePulseTestDual`

Table 13-12
KIGeneral TSP test script: ConfigPulseIMeasureVSweepLin

TSP project name:	KIFactoryPulse
TSP test script name:	KIPulse
Firmware version:	1.2.0 and later

Function:
`ConfigPulseIMeasureVSweepLin`
Usage:
`f, msg = ConfigPulseIMeasureVSweepLin (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

Description	Configures a linear pulsed current sweep with a voltage measurement at each point. Measurement(s) will be made at the end of the `ton` time. The magnitude of the first pulse will be `start` amps. The magnitude of the last pulse will be `stop` amps. The magnitude of each pulse in between will be `step` amps larger than the previous pulse where: *step = (stop - start) / (points - 1)* This function does not cause the specified `smu` to output a pulse train. It simply checks to see if all of the pulse dimensions are achievable and if so, assigns the indicated `tag` or index to it. The `InitPulseTest(tag)` and `InitPulseTestDual(tag)` functions are used to initiate a pulse train assigned to a valid `tag`.
Parameters	`smu,bias,start,stop,limit,ton,toff,points,buffer,tag [,sync_in][,sync_out])` `smu`: SourceMeter channel. (e.g. `smua`). `bias`: Pulse bias level in amps. `start`: Pulse sweep start level in amps. `stop`: Pulse sweep stop level in amps. `limit`: Voltage limit (i.e. compliance) in volts. `ton`: Pulse width (i.e. on-time) in seconds. `toff`: Pulse off time in seconds. `points`: Number of pulse measure cycles. `buffer`: Reading buffer where pulsed measurements will be stored. If this is `nil` when the function is called, no measurements will be made when the pulse train is initiated. `tag`: Numeric identifier to be assigned to the defined pulse train. `sync_in`: Defines a digital I/O trigger input line. If programmed, the pulse train will wait for a trigger input before executing each pulse. This parameter is optional. `sync_out`: Defines a digital I/O trigger output line. If programmed, the pulse train will generate a trigger output immediately prior to the start of `ton`. This parameter is optional. `sync_in_timeout`: Specifies the length of time (in seconds) to wait for input trigger (defaults to 10s). *New to Firmware 1.3.0* `sync_in_abort`: (true or false) Whether to abort pulse if input trigger is not received before timeout expires (defaults to true). *New to Firmware 1.3.0*

Table 13-12 (continued)
KIGeneral TSP test script: ConfigPulseIMeasureVSweepLin

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`ConfigPulseIMeasureVSweepLin`
Usage:
`f, msg = ConfigPulseIMeasureVSweepLin (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

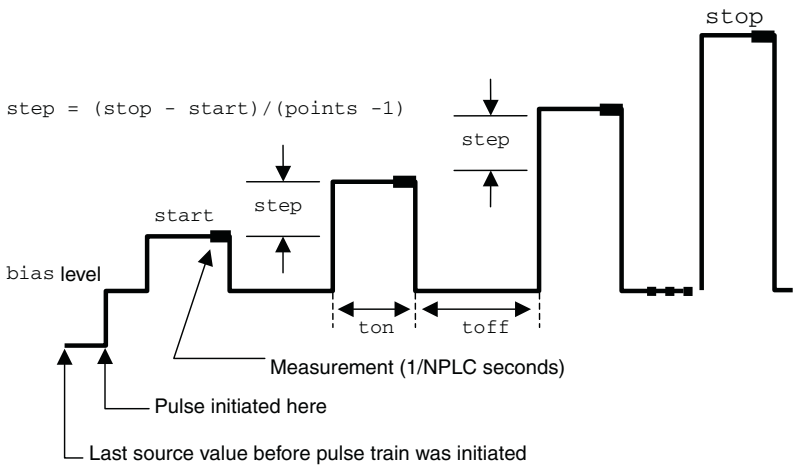
Waveform	
Return Values	`f`: A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered. `msg`: A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	Pulsed voltage measurements are stored in the reading buffer specified by the `buffer` input argument.
Example	`ConfigPulseIMeasureVSweepLin(smua, 0, 0.01, 0.05, 1, 1e-3, 0.1, 20, smua.nvbuffer2, 3)` Set up a pulsed sweep that will use SourceMeter channel A. The pulsed sweep will start at 10mA, end at 50mA, and return to a 0mA bias level between pulses. Each pulsed-step will be on for 1msec and then at the bias level for 100msec. The voltage limit will be 1V during the entire pulsed-sweep. The pulse train will be comprised of 20 pulsed-steps and the pulse train will be assigned a tag index of 4.
See Also	`InitiatePulseTest`, `InitiatePulseTestDual`

Table 13-13
KIGeneral TSP test script: ConfigPulseVMeasureISweepLin

TSP project name:	KIFactoryPulse
TSP test script name:	KIPulse
Firmware version:	1.2.0 and later

Function:
`ConfigPulseVMeasureISweepLin`
Usage:
`f, msg = ConfigPulseVMeasureISweepLin (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

Description	Configures a linear pulsed voltage sweep with a current measurement at each point. Measurement(s) will be made at the end of the `ton` time. The magnitude of the first pulse will be `start` volts. The magnitude of the last pulse will be `stop` volts. The magnitude of each pulse in between will be `step` volts larger than the previous pulse where: *step = (stop - start) / (points - 1)* This function does not cause the specified `smu` to output a pulse train. It simply checks to see if all of the pulse dimensions are achievable and if so, assigns the indicated `tag` or index to it. The `InitPulseTest(tag)` and `InitPulseTestDual(tag)` functions are used to initiate a pulse train assigned to a valid `tag`.
Parameters	`smu,bias,start,stop,limit,ton,toff,points,buffer,tag [,sync_in][,sync_out])` `smu`: SourceMeter channel. (e.g. `smua`). `bias`: Pulse bias level in volts. `start`: Pulse sweep start level in volts. `stop`: Pulse sweep stop level in volts. `limit`: Current limit (i.e. compliance) in amps. `ton`: Pulse width (i.e. on-time) in seconds. `toff`: Pulse off time in seconds. `points`: Number of pulse measure cycles. `buffer`: Reading buffer where pulsed measurements will be stored. If this is `nil` when the function is called, no measurements will be made when the pulse train is initiated. `tag`: Numeric identifier to be assigned to the defined pulse train. `sync_in`: Defines a digital I/O trigger input line. If programmed, the pulse train will wait for a trigger input before executing each pulse. This parameter is optional. `sync_out`: Defines a digital I/O trigger output line. If programmed, the pulse train will generate a trigger output immediately prior to the start of `ton`. This parameter is optional. `sync_in_timeout`: Specifies the length of time (in seconds) to wait for input trigger (defaults to 10s). *New to Firmware 1.3.0* `sync_in_abort`: (true or false) Whether to abort pulse if input trigger is not received before timeout expires (defaults to true). *New to Firmware 1.3.0*

Table 13-13 (continued)
KIGeneral TSP test script: ConfigPulseVMeasureISweepLin

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`ConfigPulseVMeasureISweepLin`
Usage:
`f, msg = ConfigPulseVMeasureISweepLin (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

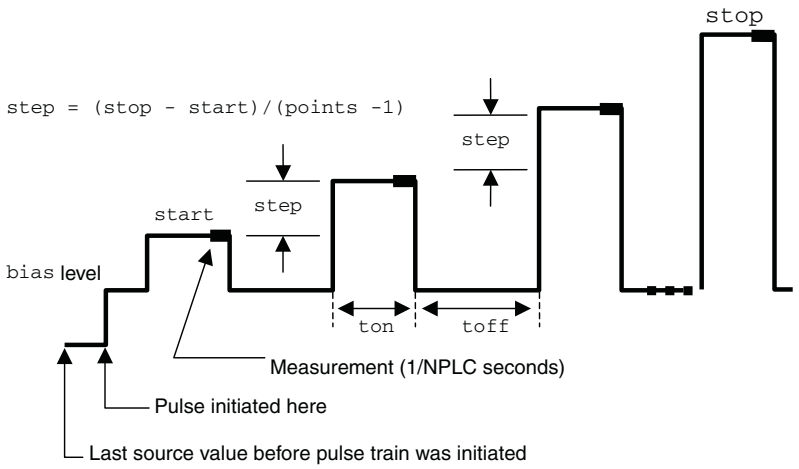
Waveform	
Return Values	`f:` A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered. `msg:` A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	Pulsed current measurements are stored in the reading buffer specified by the `buffer` input argument.
Example	`ConfigPulseVMeasureISweepLin(smub, 0, 1, 10, 1, 10e-3, 20e-3, 16, smub.nvbuffer1, 4)` Set up a pulsed sweep that will use SourceMeter channel B. The pulsed sweep will start at 1V, end at 10V, and return to a 0V bias level between pulses. Each pulsed-step will be on for 10msec and then at the bias level for 20msec. The current limit will be 1A during the entire pulsed-sweep. The pulse train will be comprised of 16 pulsed-steps and the pulse train will be assigned a tag index of 4.
See Also	`InitiatePulseTest`, `InitiatePulseTestDual`

Table 13-14
KIGeneral TSP test script: ConfigPulseIMeasureVSweepLog

TSP project name:	KIFactoryPulse
TSP test script name:	KIPulse
Firmware version:	1.2.0 and later

Function:
`ConfigPulseIMeasureVSweepLog`
Usage:
`f, msg = ConfigPulseIMeasureVSweepLog (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

Description	Configures a logarithmic pulsed current sweep with a voltage measurement at each point. Measurement(s) will be made at the end of the `ton` time. The magnitude of the first pulse will be `start` amps. The magnitude of the last pulse will be `stop` amps. The magnitude of each pulse in between will be $\texttt{LogStep}_n$ amps larger than the previous pulse where: *LogStepSize =(log10(stop) - log10(start)) / (points -1)* *LogStepn = (n - 1) x (LogStepSize) where n = [1, points]* *SourceStepLeveln = antilog(LogStepn) x start* This function does not cause the specified `smu` to output a pulse train. It simply checks to see if all of the pulse dimensions are achievable and if so, assigns the indicated `tag` or index to it. The `InitPulseTest(tag)` and `InitPulseTestDual(tag)` functions are used to initiate a pulse train assigned to a valid `tag`.
Parameters	`smu,bias,start,stop,limit,ton,toff,points,buffer,tag [,sync_in][,sync_out])` `smu`: SourceMeter channel. (e.g. `smua`). `bias`: Pulse bias level in amps. `start`: Pulse sweep start level in amps. `stop`: Pulse sweep stop level in amps. `limit`: Voltage limit (for instance compliance) in volts. `ton`: Pulse width (for instance on time) in seconds. `toff`: Pulse off time in seconds. `points`: Number of pulse measure cycles. `buffer`: Reading buffer where pulsed measurements will be stored. If this is `nil` when the function is called, no measurements will be made when the pulse train is initiated. `tag`: Numeric identifier to be assigned to the defined pulse train. `sync_in`: Defines a digital I/O trigger input line. If programmed, the pulse train will wait for a trigger input before executing each pulse. This parameter is optional. `sync_out`: Defines a digital I/O trigger output line. If programmed, the pulse train will generate a trigger output immediately prior to the start of `ton`. This parameter is optional. `sync_in_timeout`: Specifies the length of time (in seconds) to wait for input trigger (defaults to 10s). *New to Firmware 1.3.0* `sync_in_abort`: (true or false) Whether to abort pulse if input trigger is not received before timeout expires (defaults to true). *New to Firmware 1.3.0*	

Table 13-14 (continued)
KIGeneral TSP test script: ConfigPulseIMeasureVSweepLog

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`ConfigPulseIMeasureVSweepLog`
Usage:
`f, msg = ConfigPulseIMeasureVSweepLog (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

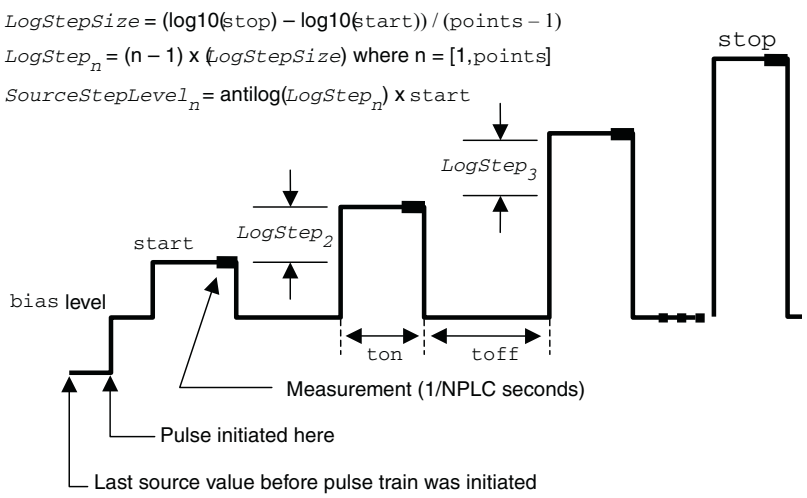
Waveform	
Return Values	`f`: A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered. `msg`: A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	Pulsed voltage measurements are stored in the reading buffer specified by the `buffer` input argument.
Example	`ConfigPulseIMeasureVSweepLog(smua, 0, 1e-3, 0.01, 1, 1e-3, 10e-3, 10, smua.nvbuffer1, 5)` Set up a pulsed log sweep that will use SourceMeter Channel A. The pulsed sweep will start at 1mA, end at 10mA, and return to a 0A bias level between pulses. Each pulsed step will be on for 1msec and then at the bias level for 10msec. The voltage limit will be 1V during the entire pulsed sweep. The pulse train will be comprised of 10 pulsed steps and the pulse train will be assigned a tag index of 5.
See Also	`InitiatePulseTest`, `InitiatePulseTestDual`

Table 13-15
KIGeneral TSP test script: ConfigPulseIMeasureVSweepLog

TSP project name:	KIFactoryPulse
TSP test script name:	KIPulse
Firmware version:	1.2.0 and later

Function:
`ConfigPulseIMeasureVSweepLog`
Usage:
`f, msg = ConfigPulseIMeasureVSweepLog (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

Description	Configures a logarithmic pulsed current sweep with a voltage measurement at each point. Measurement(s) will be made at the end of the `ton` time. The magnitude of the first pulse will be `start` amps. The magnitude of the last pulse will be `stop` amps. The magnitude of each pulse in between will be $\texttt{LogStep}_n$ amps larger than the previous pulse where: *LogStepSize =(log10(stop) - log10(start)) / (points -1)* *LogStepn = (n - 1) x (LogStepSize) where n = [1, points]* *SourceStepLeveln = antilog(LogStepn) x start* This function does not cause the specified `smu` to output a pulse train. It simply checks to see if all of the pulse dimensions are achievable and if so, assigns the indicated `tag` or index to it. The `InitPulseTest(tag)` and `InitPulseTestDual(tag)` functions are used to initiate a pulse train assigned to a valid `tag`.
Parameters	`smu,bias,start,stop,limit,ton,toff,points,buffer,tag [,sync_in][,sync_out])` `smu`: SourceMeter channel. (e.g. `smua`). `bias`: Pulse bias level in amps. `start`: Pulse sweep start level in amps. `stop`: Pulse sweep stop level in amps. `limit`: Voltage limit (i.e. compliance) in volts. `ton`: Pulse width (i.e. on-time) in seconds. `toff`: Pulse off time in seconds. `points`: Number of pulse measure cycles. `buffer`: Reading buffer where pulsed measurements will be stored. If this is `nil` when the function is called, no measurements will be made when the pulse train is initiated. `tag`: Numeric identifier to be assigned to the defined pulse train. `sync_in`: Defines a digital I/O trigger input line. If programmed, the pulse train will wait for a trigger input before executing each pulse. This parameter is optional. `sync_out`: Defines a digital I/O trigger output line. If programmed, the pulse train will generate a trigger output immediately prior to the start of `ton`. This parameter is optional. `sync_in_timeout`: Specifies the length of time (in seconds) to wait for input trigger (defaults to 10s). *New to Firmware 1.3.0* `sync_in_abort`: (true or false) Whether to abort pulse if input trigger is not received before timeout expires (defaults to true). *New to Firmware 1.3.0*

Table 13-15 (continued)
KIGeneral TSP test script: ConfigPulseIMeasureVSweepLog

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`ConfigPulseIMeasureVSweepLog`
Usage:
`f, msg = ConfigPulseIMeasureVSweepLog (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

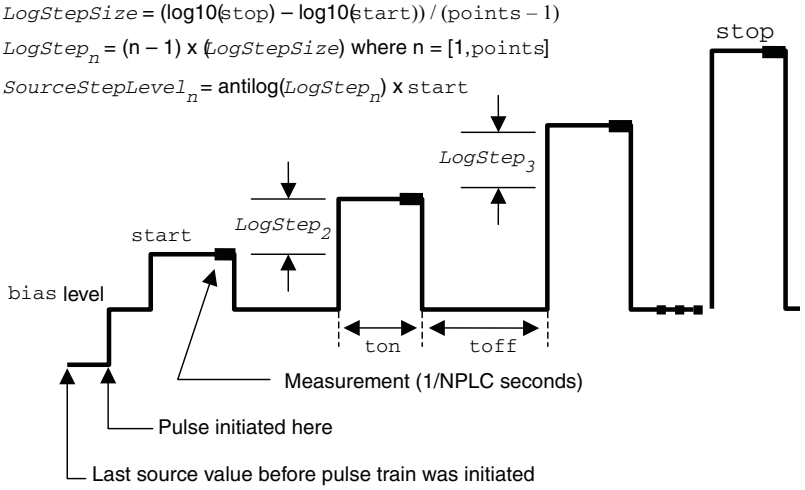
Waveform	*LogStepSize* = (log10(stop) – log10(start)) / (points – 1) $LogStep_n$ = (n – 1) x (*LogStepSize*) where n = [1,points] $SourceStepLevel_n$ = antilog($LogStep_n$) x start 
Return Values	`f`: A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered. `msg`: A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	Pulsed voltage measurements are stored in the reading buffer specified by the `buffer` input argument.
Example	`ConfigPulseIMeasureVSweepLog(smua, 0, 1e-3, 0.01, 1, 1e-3, 10e-3, 10, smua.nvbuffer1, 5)` Set up a pulsed log sweep that will use SourceMeter Channel A. The pulsed sweep will start at 1mA, end at 10mA, and return to a 0A bias level between pulses. Each pulsed-step will be on for 1msec and then at the bias level for 10msec. The voltage limit will be 1V during the entire pulsed-sweep. The pulse train will be comprised of 10 pulsed steps and the pulse train will be assigned a tag index of 5.
See Also	`InitiatePulseTest`, `InitiatePulseTestDual`

Table 13-16
KIGeneral TSP test script: ConfigPulseVMeasureISweepLog

TSP project name:	KIFactoryPulse
TSP test script name:	KIPulse
Firmware version:	1.2.0 and later

Function:
`ConfigPulseVMeasureISweepLog`
Usage:
`f, msg = ConfigPulseVMeasureISweepLog (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

Description	Configures a logarithmic pulsed voltage sweep with a current measurement at each point. Measurement(s) will be made at the end of the `ton` time. The magnitude of the first pulse will be `start` volts. The magnitude of the last pulse will be `stop` volts. The magnitude of each pulse in between will be $\texttt{LogStep}_n$ volts larger than the previous pulse where: *LogStepSize =(log10(stop) - log10(start)) / (points -1)* *LogStepn = (n - 1) x (LogStepSize) where n = [1, points]* *SourceStepLeveln = antilog(LogStepn) x start* This function does not cause the specified `smu` to output a pulse train. It simply checks to see if all of the pulse dimensions are achievable and if so, assigns the indicated `tag` or index to it. The `InitPulseTest(tag)` and `InitPulseTestDual(tag)` functions are used to initiate a pulse train assigned to a valid `tag`.
Parameters	`smu,bias,start,stop,limit,ton,toff,points,buffer,tag [,sync_in][,sync_out])` `smu`: SourceMeter channel. (e.g. `smua`). `bias`: Pulse bias level in volts. `start`: Pulse sweep start level in volts. `stop`: Pulse sweep stop level in volts. `limit`: Current limit (i.e. compliance) in amps. `ton`: Pulse width (i.e. on-time) in seconds. `toff`: Pulse off time in seconds. `points`: Number of pulse measure cycles. `buffer`: Reading buffer where pulsed measurements will be stored. If this is `nil` when the function is called, no measurements will be made when the pulse train is initiated. `tag`: Numeric identifier to be assigned to the defined pulse train. `sync_in`: Defines a digital I/O trigger input line. If programmed, the pulse train will wait for a trigger input before executing each pulse. This parameter is optional. `sync_out`: Defines a digital I/O trigger output line. If programmed, the pulse train will generate a trigger output immediately prior to the start of `ton`. This parameter is optional. `sync_in_timeout`: Specifies the length of time (in seconds) to wait for input trigger (defaults to 10s). *New to Firmware 1.3.0* `sync_in_abort`: (true or false) Whether to abort pulse if input trigger is not received before timeout expires (defaults to true). *New to Firmware 1.3.0*

Table 13-16 (continued)
KIGeneral TSP test script: ConfigPulseVMeasureISweepLog

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`ConfigPulseVMeasureISweepLog`
Usage:
`f, msg = ConfigPulseVMeasureISweepLog (smu,bias,start,stop,limit,ton,toff,points,buffer,tag[,sync_in] [,sync_out][,sync_in_timeout][,sync_in_abort])`	

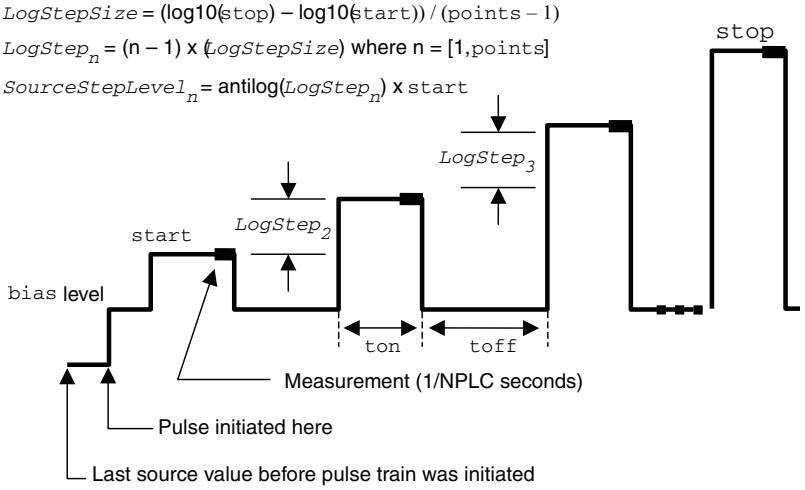
Configuration	$LogStepSize = (\log10(stop) - \log10(start)) / (points - 1)$ $LogStep_n = (n - 1) \times (LogStepSize)$ where n = [1,points] $SourceStepLevel_n = antilog(LogStep_n) \times start$ 
Return Values	`f:` A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered. `msg:` A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	Pulsed current measurements are stored in the reading buffer specified by the `buffer` input argument.
Example	`ConfigPulseVMeasureISweepLog(smub, 0, 1, 10, 1, 10e-3, 20e-3, 10, smub.nvbuffer1, 6)` Set up a pulsed log sweep that will use SourceMeter channel B. The pulsed sweep will start at 1V, end at 10V, and return to a 0V bias level between pulses. Each pulsed-step will be on for 10msec and then at the bias level for 20msec. The current limit will be 1A during the entire pulsed-sweep. The pulse train will be comprised of 10 pulsed-steps and the pulse train will be assigned a tag index of 6.
See Also	`InitiatePulseTest`, `InitiatePulseTestDual`

Table 13-17
KIGeneral TSP test script: QueryPulseConfig

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`QueryPulseConfig`
Usage:
`tbl = QueryPulseConfig(tag)`	

Description	Once a pulse train has been configured and assigned to a tag, it is often desirable to inspect the settings of this pre-configured pulse train. `QueryPulseConfig()` can be used for this purpose. This function will return a table containing the various settings associated with the tag input parameter.
Parameters	tag: Numeric identifier of pulse train configuration being queried.
Return Values	`tostring()`: A function that returns most elements in a string convenient for printing. `tag`: Identifying tag for this pulse train. `smu`: The smu configured for pulsing. `func`: Pulse function: `smuX.OUTPUT_DCAMPS` or `smuX.OUTPUT_DCVOLTS` `bias`: Pulse bias level. `level`: Pulse level for non sweeping pulses. `start`: Starting level for sweep pulses. `stop`: Ending level for sweep pulses. `limit`: Limit value. `ton`: On time in seconds. `toff`: Off time in seconds. `points`: The number of points in this pulse train. `buf`: Reference to buffer containing measurement data `sync_in`: The sync_in digio line, if used. `sync_out`: The sync_out digio line, if used. `sourcevalues`: A table containing the source value for each point in the pulse train.
Output Data	None

Table 13-17 (continued)
KIGeneral TSP test script: QueryPulseConfig

TSP project name:	KIFactoryPulse
TSP test script name:	KIPulse
Firmware version:	1.2.0 and later

Function:
`QueryPulseConfig`
Usage:
`tbl = QueryPulseConfig(tag)`	

Example(s)	`smua.reset()`
	`smua.source.rangev = 5`
	`smua.source.rangei = 1`
	`smua.source.levelv = 0`
	`smua.measure.rangev = 5`
	`smua.measure.rangei = 1`
	`smua.measure.nplc = 0.01`
	`smua.measure.autozero = smua.AUTOZERO_ONCE`
	`smua.nvbuffer1.clear()`
	`smua.nvbuffer1.appendmode = 1`
	`smua.source.output = smua.OUTPUT_ON`
	
	`f1, msg1 = ConfigPulseVMeasureI(smua, 0, 5, 1, 0.002, 0.2, 10,smua.nvbuffer1, 1)`
	
	`print(QueryPulseConfig(1).tostring())`
	
	Output from commands above
	`>> tag = 1`
	`>> smu = smua`
	`>> func = volts`
	`>> type = pulse`
	`>> bias = 0`
	`>> level = 5`
	`>> limit = 1`
	`>> time on = 0.002`
	`>> time off = 0.2`
	`>> points = 10`
	`>> measure = yes`
	`>> sync_in = 0`
	`>> sync_out = 0`
	`>> sync_in_timeout = 0`
	`>> sync_out_abort = 0`
	`>> { 5, 5, 5, 5, 5, 5, 5, 5, 5, 5 }`
	
	Configure channel A to generate a pulse train then query configuration and display as a string. Channel A will pulse voltage from a bias level of 0V to pulse level of 5V. The pulse level will be present for 2msec and the bias level for 200 msec, with a 1A limit setting. A total of 10 pulses will be generated and the measurement data will be stored in SMU A.nvbuffer1. This pulse train will be assigned to tag = 1

Table 13-17 (continued)
KIGeneral TSP test script: QueryPulseConfig

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`QueryPulseConfig`
Usage:
`tbl = QueryPulseConfig(tag)`	

See Also	`ConfigPulseIMeasureV`, `ConfigPulseVMeasureI`, `ConfigPulseIMeasureVSweepLin`, `ConfigPulseVMeasureISweepLin`, `ConfigPulseIMeasureVSweepLog`, `ConfigPulseVMeasureISweepLog`
Description	Once a pulse train has been configured and assigned to a tag, it is often times desirable to inspect the settings of this pre-configured pulse train. `QueryPulseConfig()` can be used for this purpose. This function will return a table containing the various settings associated with the tag input parameter.
For Firmware version 1.3.0	Please note that time off, sync_in_timeout, and sync_in_abort parameters will have different values.

Table 13-18
KIGeneral TSP test script: InitiatePulseTest

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`InitiatePulseTest`
Usage:
`f, msg = InitiatePulseTest(tag)`	

Description	This function initiates the pulse configuration assigned `tag`.
Parameters	tag: Numeric identifier of pulse train configuration to be initiated.
Return Values	`f`: A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered. `msg`: A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	None

Table 13-18 (continued)
KIGeneral TSP test script: InitiatePulseTest

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`InitiatePulseTest`
Usage:
`f, msg = InitiatePulseTest(tag)`	

Example(s)	`smua.reset()`
	`smua.source.rangev = 5`
	`smua.source.rangei = 1`
	`smua.source.levelv = 0`
	`smua.measure.rangev = 5`
	`smua.measure.rangei = 1`
	`smua.measure.nplc = 0.01`
	`smua.measure.autozero = smua.AUTOZERO_ONCE`
	`smua.nvbuffer1.clear()`
	`smua.nvbuffer1.appendmode = 1`
	`smua.source.output = smua.OUTPUT_ON`
	
	`f1, msg1 = ConfigPulseVMeasureI(smua, 0, 5, 1, 0.002, 0.2, 10,`
	
	`smua.nvbuffer1, 1)`
	`if (f1 == true) then`
	`  f2, msg2 = InitiatePulseTest(1)`
	`  print("Initiate message:", msg2)`
	`else`
	`  print("Config errors:", msg1)`
	`end`

Example(s)	Configure SourceMeter channels A to generate a pulse train. If no errors are encountered, initiate the pulse train. Channel A will pulse voltage from a bias level of 0V to pulse level of 5V. The pulse level will be present for 2 msec and the bias level for 200 msec, with a 1A limit setting. A total of 10 pulses will be generated and the measurement data will be stored in `smua.nvbuffer1`. This pulse train will be assigned to tag = 1.
See Also	`ConfigPulseIMeasureV`, `ConfigPulseVMeasureI`, `ConfigPulseIMeasureVSweepLin`, `ConfigPulseVMeasureISweepLin`, `ConfigPulseIMeasureVSweepLog`, `ConfigPulseVMeasureISweepLog`

Table 13-19
KIGeneral TSP test script: InitiatePulseTestDual

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`InitiatePulseTestDual`
Usage:
`f, msg = InitiatePulseTestDual(tag1, tag2)`	

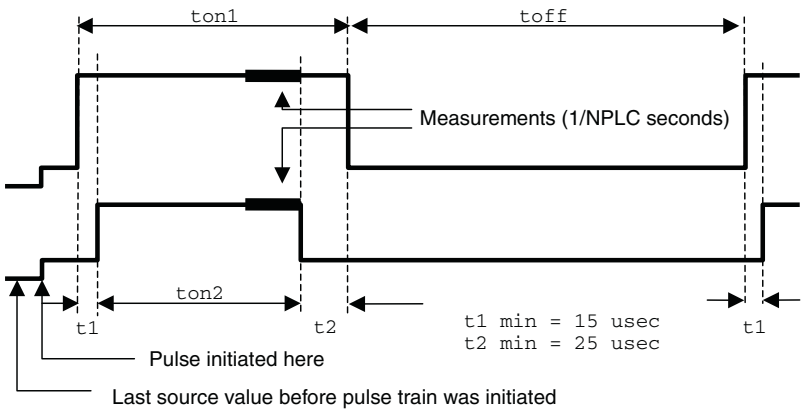
Description	This function initiates the pulse configurations assigned to `tag1` and `tag2`. The pulse trains associated with the indicated tags will be generated simultaneously. This is useful when testing devices such as voltage regulators where the input signal and output load must be applied to the device at the same time. When using this function, each `tag1` pulse will encapsulate each `tag2` pulse in time. That is, the `tag1` pulse will transition from its bias level to its pulse level before the `tag2` pulse. Both the `tag1` and `tag2` pulses will return to their respective bias levels at approximately the same time. Measurements for both pulse trains take place at the same time. (See Waveform below). To provide this encapsulation, the following rules are enforced: • The `tag1` pulse on time, `ton1`, must be configured to be at > 40usec longer than the `tag2` pulse on time. • The `tag1` and `tag2` pulse off times, `toff`, must be the same.
Waveform	
Parameters	`tag1:` Numeric identifier of the first pulse train configuration to be initiated. `tag2:` Numeric identifier of the second pulse train configuration to be initiated.
Return Values	`f:` A Boolean flag. This flag will be `true` when the pulse was successfully configured, `false` when errors were encountered. `msg:` A string message. If the `f` flag is `false`, `msg` will contain an error message. Otherwise, `msg` will contain a string indicating successful configuration.
Output Data	None

Table 13-19 (continued)
KIGeneral TSP test script: InitiatePulseTestDual

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`InitiatePulseTestDual`
Usage:
`f, msg = InitiatePulseTestDual(tag1, tag2)`	

Example(s)	`smua.reset()`
	`smua.source.rangev = 5`
	`smua.source.rangei = 1`
	`smua.source.levelv = 0`
	`smua.measure.rangev = 5`
	`smua.measure.rangei = 1`
	`smua.measure.nplc = 0.01`
	`smua.measure.autozero = smua.AUTOZERO_ONCE`
	`smua.nvbuffer1.clear()`
	`smua.nvbuffer1.appendmode = 1`
	`smua.source.output = smua.OUTPUT_ON`
	
	`smub.reset()`
	`smub.source.func = smub.OUTPUT_DCAMPS`
	`smub.source.rangei = 1`
	`smub.source.rangev = 5`
	`smub.source.leveli = 0`
	`smub.measure.rangei = 1`
	`smub.measure.rangev = 5`
	`smub.measure.nplc = 0.01`
	`smub.measure.autozero = smub.AUTOZERO_ONCE`
	`smub.nvbuffer1.clear()`
	`smub.nvbuffer1.appendmode = 1`
	`smub.source.output = smub.OUTPUT_ON`

Table 13-19 (continued)
KIGeneral TSP test script: InitiatePulseTestDual

TSP project name: KIFactoryPulse TSP test script name: KIPulse Firmware version: 1.2.0 and later	

Function:
`InitiatePulseTestDual`
Usage:
`f, msg = InitiatePulseTestDual(tag1, tag2)`	

Example(s)	`f1, msg1 = ConfigPulseVMeasureI(smua, 0, 5, 1, 0.002, 0.2, 10,smua.nvbuffer1, 1)`
	`f2, msg2 = ConfigPulseIMeasureV(smub, 0,-1, 5, 0.001, 0.2, 10,`
	
	`smub.nvbuffer1, 2)`
	`if (f1 == true) and (f2 == true) then`
	`  f3, msg3 = InitiatePulseTestDual(1,2)`
	`  print("Initiate message:", msg3)`
	`else`
	`  print("Config errors:", msg1, msg2)`
	`end`
	
	Set up SourceMeter channels A and B for pulse operation, configure pulse trains for each channel, initiate the pulse trains if no errors are encountered. Channel A will pulse voltage from a bias level of 0V to pulse level of 5V. The pulse level will be present for 2msec and the bias level for 200msec, with a 1A limit setting. A total of 10 pulses will be generated and the measurement data will be stored in SMU A.nvbuffer1. This pulse train will be assigned to tag = 1. Channel B will pulse current from a bias level of 0A to pulse level of 1A. The pulse level will be present for 1 msec and the bias level for 200msec, with a 5V limit setting. A total of 10 pulses will be generated and the measurement data will be stored in `smub.nvbuffer1`. This pulse train will be assigned to tag = 2.

See Also	`ConfigPulseIMeasureV`, `ConfigPulseVMeasureI`, `ConfigPulseIMeasureVSweepLin`, `ConfigPulseVMeasureISweepLin`, `ConfigPulseIMeasureVSweepLog`, `ConfigPulseVMeasureISweepLog`

Flash firmware upgrade

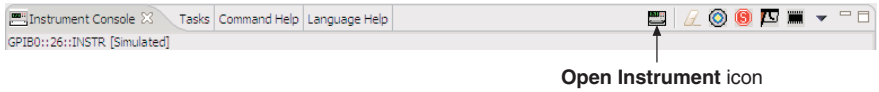
As Keithley Instruments develops additional factory scripts, they will be made available on the Keithley Instruments website (www.keithley.com) as a flash firmware upgrade for the Series 2600.

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 Instruments website, use the Test Script Builder (TSB) to upgrade the firmware of your Series 2600:

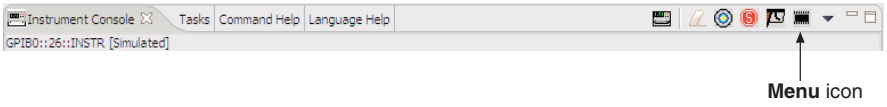
1. On the PC desktop, double-click the icon for the Test Script Builder.
2. On the Instrument Console toolbar, click the Open Instrument icon and then select your communication interface from the Select Instrument Resource dialog box. Details on opening communications are provided in Figure 2-8.

Figure 13-1
Pulse sweep example



3. On the Instrument Console toolbar, click the Menu icon to display the menu.

Figure 13-2
Pulse sweep example



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 Series 2600.

This page left blank intentionally.

Section 14

Display Operations

In this section:

Topic	Page
Display functions and attributes	**14-2**
Display features	**14-2**
Display screen	14-2
Measurement functions	14-3
Display resolution	14-3
Input prompting	**14-7**
Menu	14-7
Parameter value prompting	14-8
Annunciators	**14-9**
LOCAL lockout	**14-10**
Load test menu	**14-10**
Saving a user script	14-11
Adding USER TESTS menu entries	14-11
Deleting USER TESTS menu entries	14-12
Running a test from the front panel	14-12
Display triggering	**14-12**
Key-press codes	**14-13**
Sending keycodes	14-13
Capturing key-press codes	14-13

Display functions and attributes

The display functions and attributes are used to perform the display operations covered in this section. Table 14-1 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.

Table 14-1
Cross referencing functions/attributes to section topics

Function/Attribute	Section Topic
`display.clear`	Clearing the display
`display.getannunciators`	Annunciators
`display.getcursor`	Cursor position
`display.gettext`	Displaying text messages
`display.input`	Capturing key-press codes
`display.inputvalue`	Parameter value prompting
`display.loadmenu.add` `display.loadmenu.delete`	Load test menu
`display.locallockout`	LOCAL lockout
`display.menu`	Menu
`display.prompt`	Parameter value prompting
`display.screen`	Display screen
`display.sendkey`	Sending keycodes
`display.setcursor`	Cursor position
`display.settext`	Displaying text messages
`display.smuX.digits`	Display resolution
`display.smuX.measure.func`	Measurement functions
`display.trigger.clear` `display.trigger.wait`	Display triggering

Display features

Display screen

The Keithley Instruments Series 2600 System SourceMeter® can display source-measure values and 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 (Models 2602/2612/2636 only).
- Source-measure screen – Display source values and measure readings for SMU A and SMU B (Models 2602/2612/2636 only).
- 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` = `smua` 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.smuX.digits` 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

NOTE Most of the display functions and attributes that are associated with display messaging will automatically select the user screen. The attribute for the display screen is explained in "Display screen" in this section.

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

The display of the Series 2600 can be used to display user-defined messages. For example, while a test is running, the following message can be displayed on the Series 2600.

```
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:

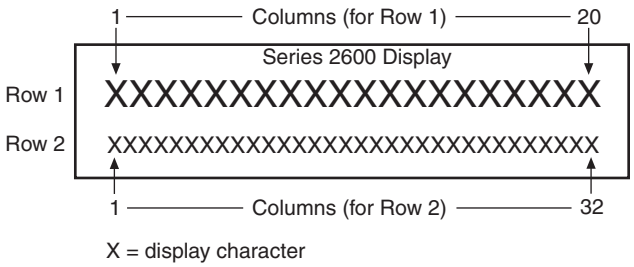
```
display.clear()
```

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 embedded in the `text` string to configure and customize the message:

`$N` Newline – Starts text on the next line. If the cursor is already on line 2, text will be ignored after the ‘`$N`’ is received.

`$R` Sets text to Normal.

`$B` Sets text to Blink.

`$D` Sets text to Dim intensity.

`$F` Set text to background blink.

`$$` Escape sequence to display a single “$”.

In addition to displaying alpha-numeric characters, other special characters can be displayed. Refer to Appendix F for a compete listing of special characters and their corresponding codes. For example, to display the Greek symbol omega, Ω , use the following:

```
display.clear()
c = string.char(18)
display.settext(c)
```

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 is 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 it's 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 Defines the digit positions for the value. Up to six zeros (0) 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 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 in 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`.

NOTE If the operator presses EXIT instead of entering a source value, `value` will be set to `nil`.

The second line of the above code can be replaced using the other input field function:

```
value = display.inputvalue("0.00")
```

The only difference is that the display prompt will not include the "V" units designator and the "Enter source value" message.

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.

Table 14-2 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 Table 14-2, the following annunciators are on: REL, REM, EDIT, AUTO, 4W and EDIT.

Table 14-2
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 Table 14-2 are used by the Series 2600.

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 Series 2600, it is not automatically added to the front panel USER TESTS submenu. A menu name and a chunk is added by the user (see "Adding USER TESTS menu entries" below).

Saving a user script

After a user script is loaded into the Series 2600 it can be saved in non-volatile memory. If it is not stored in non-volatile memory, the script will be lost when the Series 2600 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" in Section 2).

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" in Section 2).

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)
```

`displayname`	Name string to add to the menu.
`chunk`	Chunk is the code to be executed.
`memory`	Save or don't save `chunk` and `displayname` 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 Series 2600 is turned off. See **Example 1** below.

Example 1:

Assume a script with a function named "DUT1" has already been loaded into the Series 2600, 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 Series 2600. 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 and press ENTER or the wheel.
4. Press the RUN key to run the test.

Display triggering

Script execution can be delayed 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 **T**RIG 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 Table 14-3.

Key press example:

Either of the following commands will press the **MENU** key:

```
display.sendkey(display.KEY_MENU)
display.sendkey(68)
```

Table 14-3
Keycodes to send for display.sendkey

display.KEY_AUTO or 73	display.KEY_OUTPUTA or 88
display.KEY_CONFIG or 80	display.KEY_OUTPUTB or 96
display.KEY_DIGITSA or 87	display.KEY_RANGEDOWN or 81
display.KEY_DIGITSB or 84	display.KEY_RANGEUP or 65
display.KEY_DISPLAY or 72	display.KEY_RECALL or 85
display.KEY_ENTER or 82	display.KEY_RELA or 70
display.KEY_EXIT or 75	display.KEY_RELB or 67
display.KEY_FILTERA or 77	display.KEY_RIGHT or 103
display.KEY_FILTERB or 74	display.KEY_RUN or 71
display.KEY_LEFT or 104	display.KEY_SPEEDA or 94
display.KEY_LIMITA or 93	display.KEY_SPEEDB or 91
display.KEY_LIMITB or 90	display.KEY_SRCA or 79
display.KEY_LOAD or 95	display.KEY_SRCB or 76
display.KEY_MEASA or 86	display.KEY_STORE or 78
display.KEY_MEASB or 83	display.KEY_TRIG or 92
display.KEY_MENU or 68	display.WHEEL_ENTER or 97
display.KEY_MODEA or 69	display.WHEEL_LEFT or 107
display.KEY_MODEB or 66	display.WHEEL_RIGHT or 114

Capturing key-press codes

A history of the keycode for the last pressed front panel key is maintained by the Series 2600. 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 Table 14-4). Keep in mind that a value of `0` (`display.KEY_NONE`) indicates that the keycode history had been cleared.

Table 14-4
Keycode values returned for display.getlastkey

0 (display.KEY_NONE)	82 (display.KEY_ENTER)
65 (display.KEY_RANGEUP)	83 (display.KEY_MEASB)
67 (display.KEY_RELB)	84 (display.KEY_DIGITSB)
68 (display.KEY_MENU)	85 (display.KEY_RECALL)
69 (display.KEY_MODEA)	86 (display.KEY_MEASA)
70 (display.KEY_RELA)	87 (display.KEY_DIGITSA)
71 (display.KEY_RUN)	90 (display.KEY_LIMITB)
72 (display.KEY_DISPLAY)	91 (display.KEY_SPEEDB)
73 (display.KEY_AUTO)	92 (display.KEY_TRIG)
74 (display.KEY_FILTERB)	93 (display.KEY_LIMITA)
75 (display.KEY_EXIT)	94 (display.KEY_SPEEDA)
76 (display.KEY_SRCB)	95 (display.KEY_LOAD)
77 (display.KEY_FILTERA)	97 (display.WHEEL_ENTER)
78 (display.KEY_STORE)	103 (display.KEY_RIGHT)
79 (display.KEY_SRCA)	104 (display.KEY_LEFT)
80 (display.KEY_CONFIG)	114 (display.WHEEL_RIGHT)
81 (display.KEY_RANGEDOWN)	

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 Table 14-3.

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.

This page left blank intentionally.

Section 15

Performance Verification

In this section:

Topic	Page
Introduction	**15-2**
Verification test requirements	**15-2**
Environmental conditions	15-2
Warm-up period	15-2
Line power	15-3
Recommended test equipment	15-3
Verification limits	15-3
Restoring factory defaults	**15-4**
Performing the verification test procedures	**15-5**
Test summary	15-5
Test considerations	15-5
Setting the source range and output value	15-5
Setting the measurement range	15-6
Output voltage accuracy	**15-6**
Voltage measurement accuracy	**15-8**
Output current accuracy	**15-9**
Model 2635/2636 output current accuracy 1nA to 100nA ranges	15-11
Current measurement accuracy	**15-10**
Series 2600 current measurement accuracy 100nA and higher	15-11
Model 2635/2636 current measurement accuracy 100pA to 100nA ranges	15-15

Introduction

Use the procedures in this section to verify that the Keithley Instruments Series 2600 System SourceMeter® 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 Instruments representative or the factory to determine the correct course of action.

Verification test requirements

Be sure that you perform the verification tests:

- Under the proper environmental conditions.
- After the specified warm-up period.
- Using the correct line voltage.
- Using the proper test equipment.
- Using the specified output signal and reading limits.

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 Series 2600 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 Series 2600 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 to 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 Series 2600 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[1] (2601/2602)	90mV: 0.9V: 5.4V: 36V:	±8ppm ±5ppm ±4ppm ±6ppm
		DC Voltage[2] (2611/2612)	190mV: 1.8V: 18V: 180V:	±5ppm ±4ppm ±6ppm ±6ppm
		DC current[3]	90nA: 0.9µA: 9µA: 90µA: 0.9mA: 9mA: 90mA: 0.9A:	±430ppm ±45ppm ±25ppm ±23ppm ±20ppm ±20ppm ±35ppm ±110ppm
0.5Ω , 250W, 0.1% Precision Resistor	Isotek RUG-Z-R500-0.1-TK3	Resistance[4]	0.5Ω :	±125ppm
1GΩ , 200V, 1% standard	Keithley 2600-STD-RES	Resistance[5]	1GΩ :	250ppm

1. 90-day specifications show full-range accuracy of recommended model used for specified measurement point.
2. Id.
3. Id.
4. Resistor used to test Model 2601/2602 3A range and Model 2611/2612/2635/2636 1.5A range only should be characterized to uncertainty shown using resistance function of digital multimeter before use.
5. Standard is a guarded and characterized 1 Gohm resistor used to test Model 2635/2636 100pA to 100nA current ranges.

Verification limits

The verification limits stated in this section have been calculated using only the Series 2600 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 Series 2600 specifications and corresponding test equipment specifications.

Example limits calculations

Model 2601/2602 example

As an example of how verification limits are calculated, assume you are testing the Model 2601/2602 6V DC output range using a 5.4V output value. Using the Model 2601/2602 one-year accuracy specification for 5.4V DC output of ± (0.02% of output + 1.8mV offset), the calculated output limits are:

Output limits = 5.4V ± [(5.4V × 0.02%) + 1.8mV]
Output limits = 5.4V ± (0.00108 + 0.0018)
Output limits = 5.4V ± 0.00288V
Output limits = 5.39712V to 5.40288V

Model 2611/2612/2635/2636 example

Similarly, assume you are testing the Model 2611/2612/2635/2636 20V DC output range using an18V output value. Using the Model 2611/2612/2635/2636 one-year accuracy specification for 18V DC output of ± (0.02% of output + 5mV offset), the calculated output limits are:

Output limits = 18V ± [(18V × 0.02%) + 5mV]
Output limits = 18V ± (0.0036 + 0.005)
Output limits = 18V ± 0.0086V
Output limits = 17.9914V to 18.0086V

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 Series 2600 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 Series 2600 output terminals (use 4-wire sensing for voltage).
- Make sure the Series 2600 is set to the correct source range.
- Be sure the Series 2600 output is turned on before making measurements.
- Be sure the test equipment is set up for the proper function and range.
- Allow the Series 2600 output signal to settle before making a measurement.
- Do not connect test equipment to the Series 2600 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 break down in insulation, creating a shock hazard.***
The Input/Output terminals of the SourceMeters are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500V peak. Do not connect the Series 2600 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 Series 2600 while the unit is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Series 2600 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 Navigation Wheel 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.
4. Use the Navigation Wheel and CURSOR keys to set the source value to the required value, then press ENTER or the Navigation Wheel to complete editing.

Setting the measurement range

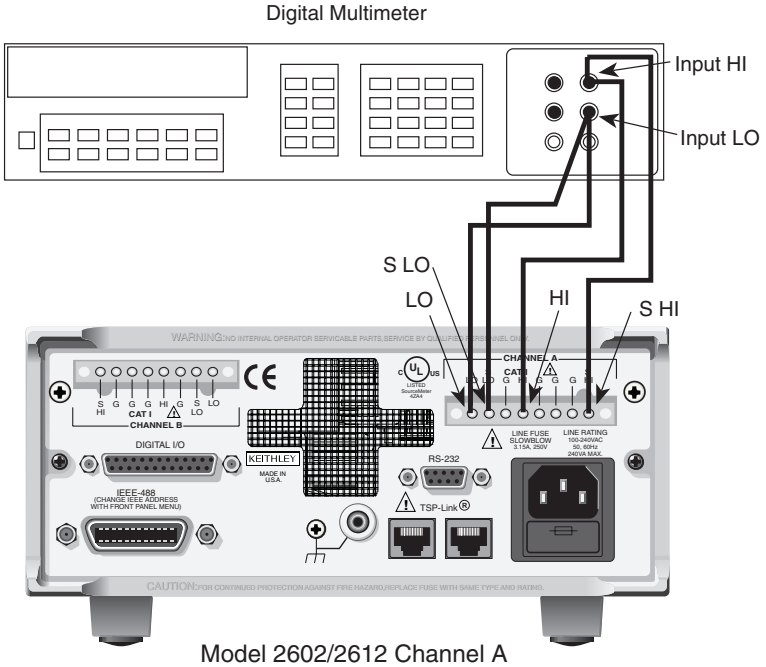
When simultaneously sourcing and measuring either voltage or current, the measure range is coupled to the source range, and you cannot independently control the measure range. Thus, it is not necessary for you to set the range when testing voltage or current measurement accuracy.

Output voltage accuracy

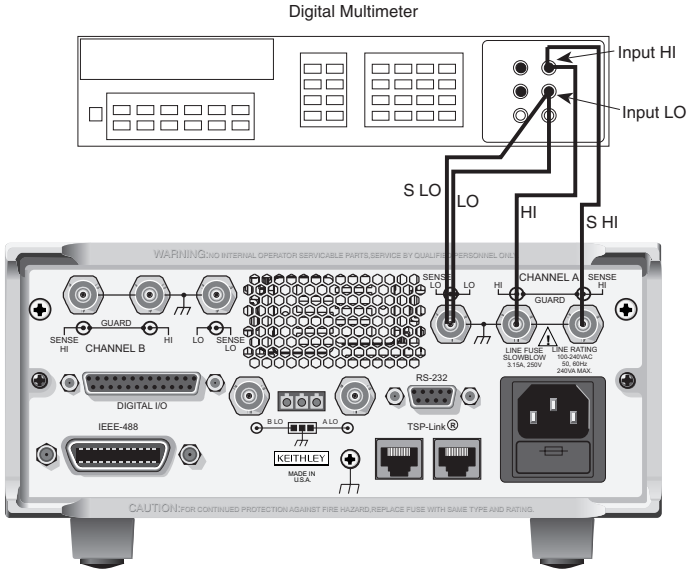
Follow the steps below to verify that the Series 2600 output voltage accuracy is within specified limits. This test involves setting the output voltage to each full-range value and measuring the voltages with a precision digital multimeter.

1. With the power off, connect the digital multimeter to the Series 2600 output terminals using 4-wire connections, as shown in Figure 15-1.

Figure 15-1
Connections for voltage verification



Model 2602/2612 Channel A Connections Shown



Model 2636 channel A connections shown

2. Select the multimeter DC volts measuring function.
3. Select the Model 2602/2612/2636 single-channel display mode. Press the Series 2600 SRC key to source voltage, and make sure the source output is turned on.
4. Enable the Series 2600 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 (Model 2601/2602) or Table 15-3 (Model 2611/2612/2635/2636). For each test point:
   - Select the correct source range.

- Set the Series 2600 output voltage to the indicated value.
- Verify that the multimeter reading is within the limits given in the table.

1. Repeat the procedure for negative output voltages with the same magnitudes as those listed in Table 15-2 or Table 15-3.

Table 15-2
Model 2601/2602 output voltage accuracy limits

Model 2601/2602 source range	Model 2601/2602 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

Table 15-3
Model 2611/2612/2635/2636 output voltage accuracy limits

Model 2611/2612/2635/2636 source range	Model 2611/2612/2635/2636 output voltage setting	Output voltage limits (1 year, 18°C–28°C)
200mV	180.000mV	179.589 to 180.411mV
2V	1.80000V	1.79904 to 1.80096V
20V	18.000V	17.9914 to 18.0086V
200V	180.000V	179.914 to 180.086V

2. For the Model 2602/2612/2636, repeat the above procedure for the other channel.

Voltage measurement accuracy

Follow the steps below to verify that the Series 2600 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 Series 2600 voltage readings are within required limits.

1. With the power off, connect the digital multimeter to the Series 2600 output terminals using 4-wire connections, as shown in Figure 15-1.
2. Select the multimeter DC volts function.
3. Select the Model 2602/2612/2636 single-channel display mode.
4. Enable the Series 2600 4-wire (remote sense) mode by pressing CONFIG then MEAS, then select V-MEAS > SENSE-MODE > 4-WIRE.
5. Set the Series 2600 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 15-4 (Model 2601/2602) or Table 15-5 (Model 2611/2612/2635/2636). For each test point:
   - Select the correct source range.
   - Set the Series 2600 output voltage to the indicated value **as measured by the digital multimeter**.
   - If necessary, press the TRIG key to display readings.
   - Verify that the Series 2600 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 15-4 or Table 15-5.
8. For the Model 2602/2612/2636, repeat the above procedure for the other channel.

Table 15-4
Model 2601/2602 voltage measurement accuracy limits

Model 2601/2602 source and measure range[1]	Source voltage[2]	Model 2601/2602 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.

Table 15-5
Model 2611/2612/2635/2636 voltage measurement accuracy limits

Model 2611/2612/2635/2636 source and measure range[1]	Source voltage[2]	Model 2611/2612/2635/2636 voltage reading limits (1 year, 18°C–28°C)
200mV	180.000mV	179.748 to 180.252mV
2V	1.80000V	1.79929 to 1.80071V
20V	18.0000V	17.9923 to 18,0077V
200V	180.000V	179.923 to 180.077V

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

Series 2600 output current accuracy 100nA and higher

Follow the steps below to verify that the Series 2600 output current accuracy is within specified limits.

NOTE An alternate procedure for 100nA current accuracy is shown in the 1nA to 100nA Output current accuracy procedure for the 2635/2636.

1. With the power off, connect the digital multimeter to the Series 2600 output terminals, as shown in Figure 15-2.
2. Select the multimeter DC current measuring function.
3. Select the Model 2602/2612/2636 single-channel display mode.
4. Press the Series 2600 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 1uA to 1A ranges listed in Table 15-6 (Model 2601/2602), or Table 15-7 (Model 2611/2612), or Table 15-8 (Model 2635/2636). For each test point:
   - Select the correct source range.
   - Set the Series 2600 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 15-6, or Table 15-7, or Table 15-8.

Table 15-6
Model 2601/2602 output current accuracy limits

Model 2601/2602 source range	Model 2601/2602 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	0.89865 to 0.90135A
3A	2.40000A	2.39706 to 2.40294A

Table 15-7
Model 2611/2612 output current accuracy limits

Model 2611/2612 source range	Model 2611/2612 output current setting	Output current limits (1 year, 18°C–28°C)
100nA	90.000nA	89.846 to 90.154nA
1µA	0.90000µA	0.89893 to 0.90107µA
10µA	9.0000µA	8.9923 to 9.0077µA
100µA	90.000µA	89.913 to 90.087µA
1mA	0.90000mA	0.89943 to 0.90057mA
10mA	9.0000mA	8.9913 to 9.0087mA
100mA	90.000mA	89.943 to 90.057mA
1A	0.90000A	0.89775 to 0.90225A
1.5A	1.35000A	1.34519 to 1.35481A

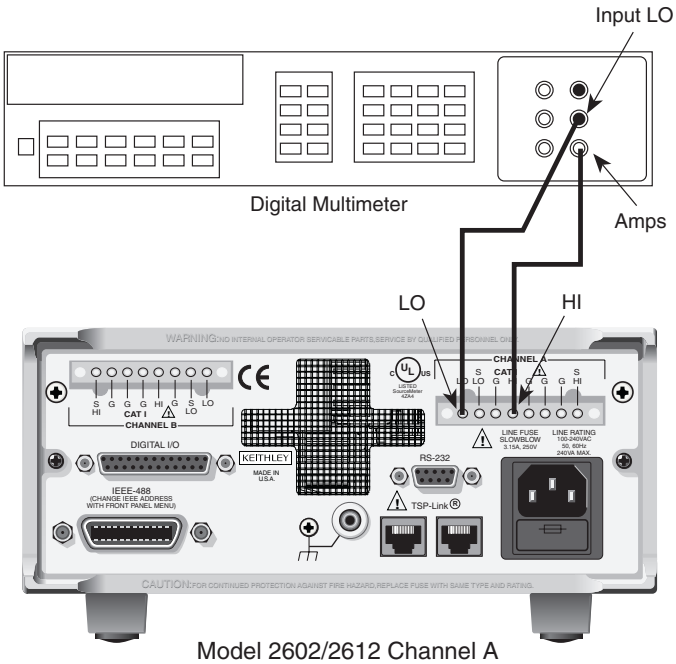
7. Turn the output off, and change connections as shown in Figure 15-3 (use 4-wire connections to the 0.5Ω resistor as shown).
8. Select the DMM DC volts function.
9. Repeat steps 4 through 7 for the 3A range (Model 2601/2602) or 1.5A range (Model 2611/2612/2635/2636). Calculate the current from the DMM voltage reading and characterized 0.5Ω resistance value: I=V/R.
10. For the Model 2602/2612/2636, repeat the above procedure for the other channel.

Model 2635/2636 output current accuracy 1nA to 100nA ranges

A suitably guarded and characterized 1 Gohm resistance standard, such as Keithley's 2600-STD-RES is necessary for the following measurements. Step-by-step procedures and connection diagrams for verifying the output current accuracy for the low current ranges are included with the 2600-STD-RES. The general process entails measuring the voltage across the characterized 1Gohm resistor for a given output current and comparing the derived current to the current accuracy of Table 15-8 for each current range.

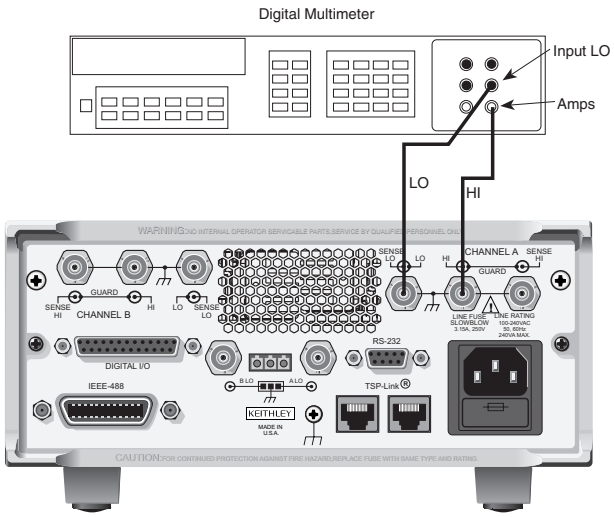
1. Connect the guarded resistance standard to the 2635/2636 and the DMM.
2. Source the appropriate current for +/- full scale reading.
3. Wait 30 seconds for stable measurement.
4. Capture the reported voltage measurement.
5. Calculate the current from measured voltage and characterized resistance.
6. Verify output current accuracy for each of the currents for the 1nA to 100nA ranges listed in Table 15-8 (Model 2635/2636).

Figure 15-2
Current verification connections (2602/2612(3A); 2636(1.5A))



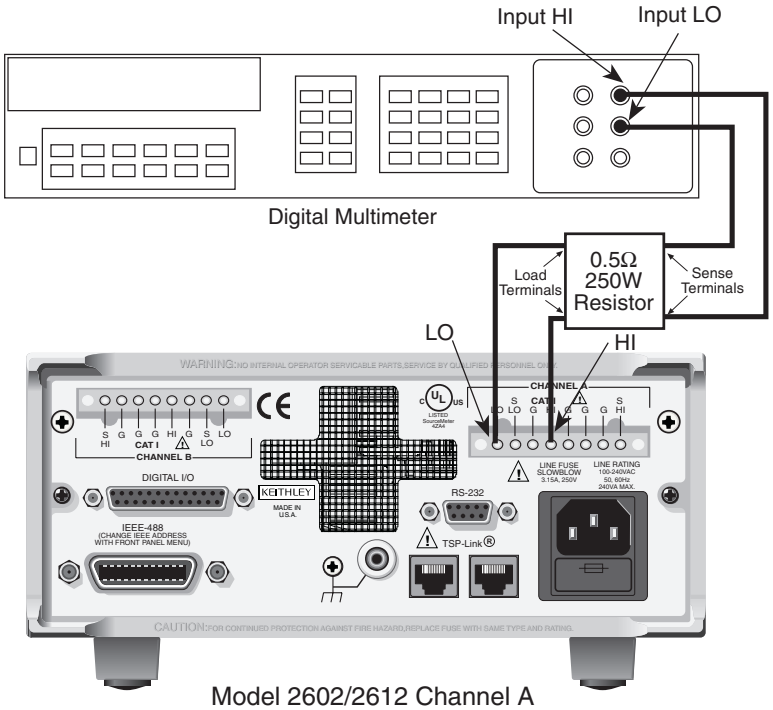
Model 2602/2612 Channel A Connections Shown

Current verification connections (1uA to 1A ranges)

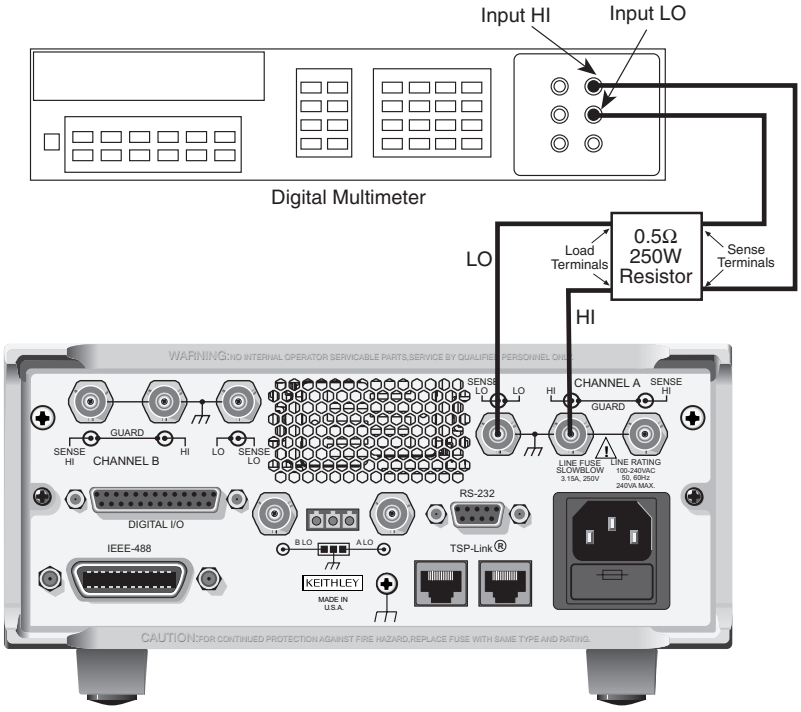


Model 2636 channel A connections shown

Figure 15-3
Current verification connection ranges (2601/2602 (3A); 2611/2612/2635/2636 (1.5A))



Model 2602/2612 Channel A Connections Shown



Model 2636 channel A connections shown

Table 15-8
Model 2635/2636 output current accuracy limits

2635/2636 Source range	2635/2636 output current setting	Output current limits (1 year 18C - 28C)
1nA	0.90000nA	0.89665 to 0.90335nA
10nA	9.0000nA	8.9815 to 9.0185nA
100nA	90.000nA	89.8960 to 90.0140nA
1uA	0.90000uA	0.89903 to 0.90097uA
10uA	9.0000uA	8.9923 to 9.0077uA
100uA	90.000uA	89.913 to 90.087uA
1mA	0.90000mA	0.89943 to 0.90057mA
10mA	9.0000mA	8.9913 to 9.0087mA
100mA	90.000mA	89.943 to 90.057mA
1A	0.90000A	0.89775 to 0.90225A
1.5A	1.35000A	1.34519 to 1.35481A

Current measurement accuracy

Series 2600 current measurement accuracy 100nA and higher

Follow the steps below to verify that Series 2600 current measurement accuracy is within specified limits. The procedure involves applying accurate currents from the Series 2600 current source and then verifying that Series 2600 current measurements are within required limits.

1. With the power off, connect the digital multimeter to the Series 2600 terminals as shown in Figure 15-2.
2. Select the multimeter DC current function.
3. Select the Model 2602/2612/2636 single-channel display mode.
4. Set the Series 2600 to both source and measure current by pressing the SRC and MEAS keys, and make sure the source output is turned on.
5. Verify measure current accuracy for each of the currents listed in Table 15-10 (Model 2601/2602), or Table 15-11 (Model 2611/2612), or Table 15-12 (Model 2635/2636). For each measurement:
   - Select the correct source range.
   - Set the Series 2600 source output to the correct value as measured by the digital multimeter.
   - If necessary, press the TRIG key to display readings.
   - Verify that the Series 2600 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 15-10, or Table 15-11, or Table 15-12.
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 (Model 2601/2602) or 1.5A range (Model 2611/2612/2635/2636). Calculate the current from the DMM voltage reading and characterized 0.5Ω resistance value.
9. For the Model 2602/2612/2636, repeat the above procedure for the other channel.

Model 2635/2636 current measurement accuracy 100pA to 100nA ranges

A suitably guarded and characterized 1 GΩ resistance standard, such as Keithley's 2600-STD-RES is necessary for the following measurements. Step-by-step procedures and connection diagrams for verifying the current measurement accuracy for the low current ranges are included with the 2600-STD-RES. The general process entails forcing a characterized voltage across the 1Gohm resistor and comparing the 2636/2636 measured results against the standard resistance and voltage derived current.

1. Characterize the appropriate +/- V source values with the DMM according to Table 15-9.

Table 15-9
Model 2635/2636 Characterization of Voltage Source settings

Low Current Range	Voltage Source	Compliance
100 pA	+/- 100.00 mV	1.5 A
1 nA	+/- 1.0000 V	1.5 A
10 nA	+/- 10.000 V	1.5 A
100 nA	+/- 100.00 V	100 mA

2. Characterize the desired 2635/2636 current ranges.
   a. Connect guarded resistance standard.
   b. Source the appropriate voltage for +/- full scale reading.
   c. Wait 30 seconds for stable measurement.
   d. Capture the 2635/2636 reported current measurement.
   e. Verify output current accuracy for each of the currents for the 100pA to 100nA ranges listed in Table 15-12 (Model 2635/2636).

Table 15-10
Model 2601/2602 current measurement accuracy limits

Model 2601/2602 source and measure range[1]	Source current[2]	Model 2601/2602 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	8.99625 to 9.00375µA
100µA	90.000µA	89.957 to 90.043µA
1mA	0.9000mA	0.89962 to 0.90038mA
10mA	9.0000mA	8.9957 to 9.0043mA
100mA	90.000mA	89.962 to 90.038mA
1A	0.90000A	0.89823 to 0.90177A
3A	2.4000A	2.3953 to 2.4047A

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.

Table 15-11
Model 2611/2612 current measurement accuracy limits

Model 2611/2612 source and measure range[1]	Source current[2]	Model 2611/2612 current reading limits (1 year, 18□C–28□C)
100nA	90.000nA	89.855 to 90.145nA
1µA	0.9000µA	0.899275 to 0.900725µA
10µA	9.0000µA	8.99625 to 9.00375µA
100µA	90.000µA	89.957 to 90.043µA
1mA	0.9000mA	0.89962 to 0.90038mA
10mA	9.0000mA	8.9957 to 9.0043mA
100mA	90.000mA	89.962 to 90.038mA
1A	0.90000A	0.89823 to 0.90177A
1.5A	1.3500A	1.345825 to 1.354175A

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.

Table 15-12
Model 2635/2636 current measurement accuracy limits

2635/2636 Source and measure range[a]	35/36 Source current[b]	Current reading limits (1 year 18C - 28C)
100pA	90.000pA	89.7850 to 90.2150pA
1nA	0.90000nA	0.89841 to 0.90159nA
10nA	9.0000nA	8.9835 to 9.0165nA
100nA	90.000nA	89.9060 to 90.0940nA
1uA	0.90000uA	0.899375 to 0.900625uA
10uA	9.0000uA	8.99625 to 9.00375uA
100uA	90.000uA	89.957 to 90.043uA
1mA	0.90000mA	0.89962 to 0.90038mA
10mA	9.0000mA	8.9957 to 9.0043mA
100mA	90.000mA	89.962 to 90.038mA
1A	0.90000A	0.89823 to 0.90177A
1.5A	1.35000A	1.345825 to 1.354175A

a. Measure range coupled to source range when simultaneously sourcing and measuring current.
b. As measured by precision digital multimeter. Use closest possible value, and modify reading limits accordingly if necessary.

Section 16

Calibration

In this section:

Topic	Page
Introduction	**16-2**
Environmental conditions	**16-2**
Temperature and relative humidity	16-2
Warm-up period	16-2
Line power	16-2
Calibration considerations	**16-2**
Calibration cycle	16-3
Recommended calibration equipment	16-3
Calibration errors	16-5
Calibration	**16-5**
Calibration steps	16-5
Calibration commands	16-7
Calibration procedure	16-8

Introduction

Use the procedures in this section to calibrate the Keithley Instruments Series 2600 System SourceMeter® (Models 2601, 2602, 2611, 2635, and 2636). These 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°C to 28°C (65°F to 82°F), with relative humidity of less than 70% (unless otherwise noted).

Warm-up period

Allow the Series 2600 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 Series 2600 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 2600 terminals.
- Always allow the source signal to settle before calibrating each point.
- Do not connect test equipment to the Series 2600 through a scanner or other switching equipment.
- If an error occurs during calibration, the Series 2600 will generate an appropriate error message. See “"Error summary"” in Appendix B 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 that could result in personal injury or death.***

The Input/Output terminals of the SourceMeters are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500V peak. Do not connect the Series 2600 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 Series 2600 while the unit is on. Power off the equipment from the front panel or disconnect the main power cord from the rear of the Series 2600 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 Series 2600 specifications.

Table 16-1
Recommended calibration equipment

Description	Manufacturer/Model	Accuracy		
1GΩ, 200V, 1% standard	Keithley Instruments Model 2600-STD-RES	Resistance[6]	1GΩ:	XXXPPM
Digital Multimeter	Agilent 3458A	DC Voltage[1] (2601/2602)	90mV: 0.9V: 5.4V: 36V:	±8ppm ±5ppm ±4ppm ±6ppm
		DC Voltage[2] (2611/2612)	190mV: 1.8V: 18V: 180V:	±5ppm ±4ppm ±6ppm ±6ppm
		DC current[3]	90nA: 0.9µA: 9µA: 90µA: 0.9mA: 9mA: 90mA: 0.9A:	±430ppm ±45ppm ±25ppm ±23ppm ±20ppm ±20ppm ±35ppm ±110ppm
0.5Ω , 250W, 0.1% Precision Resistor	Isotek RUG-Z-R500-0.1-TK3	Resistance[4]	0.5Ω :	±125ppm
50Ω Resistors (2)	Any suitable.[5]			
1GΩ , 200V, 1% standard	Keithley 2600-STD-RES	Resistance[6]	1GΩ :	+/-250ppm

1. 90-day specifications show full-range accuracy of recommended model used for specified calibration point.
2. Id.
3. Id.
4. Resistor used to calibrate Model 2601/2602 3A and 10A ranges and Model 2611/2612/2635/2636 1.5A and 10A ranges should be characterized to uncertainty shown using resistance function of a digital multimeter before use.
5. Used for contact check calibration. Characterize resistors using ohms function of digital multimeter before use.
6. Standard is a guarded and characterized 1GΩ resistor used to test Model 2635/2636 100pA to 100nA current ranges.

Calibration errors

The Series 2600 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 Series 2600 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 (Model 2601/2602), or Table 16-3 (Model 2611/2612), or Table 16-5 (Model 2635/2636). 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.

Table 16-2
Model 2601/2602 calibration steps

Function[1]	Calibration steps[2]	Calibration points[4]	Sense mode[5]
Voltage Source and Measure	100mV	±1e-12, ±90mV	smuX.SENSE_LOCAL
	100mV	±1e-10, ±90mV	smuX.SENSE_REMOTE
	1V	±1e-10, ±0.9V	smuX.SENSE_LOCAL
	1V	±1e-10, ±0.9V	smuX.SENSE_CALA
	6V	±1e-10, ±5.4V	smuX.SENSE_LOCAL
	40V	±1e-10, ±36V	smuX.SENSE_LOCAL
Current Source and Measure	100nA	±1e-10, ±90nA	smuX.SENSE_LOCAL
	1µA	±1e-10, ±0.9µA	smuX.SENSE_LOCAL
	10µA	±1e-10, ±9µA	smuX.SENSE_LOCAL
	100µA	±1e-10, ±90µA	smuX.SENSE_LOCAL
	1mA	±1e-10, ±0.9mA	smuX.SENSE_LOCAL
	1mA	±1e-10, ±0.9mA	smuX.SENSE_CALA
	10mA	±1e-10, ±9mA	smuX.SENSE_LOCAL
	100mA	±1e-10, ±90mA	smuX.SENSE_LOCAL
	1A	±1e-10, ±0.9A	smuX.SENSE_LOCAL
	3A	±1e-10, ±2.4A	smuX.SENSE_LOCAL
	10A[3]	±1e-10, ±2.4A	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 ±1e-10. Calibration polarities must also be set as shown in the procedures.
5. Output must be off before changing to the CALA sense mode.

Table 16-3
Model 2611/2612 calibration steps

Function[1]	Calibration steps[2]	Calibration points[3]	Sense mode[4]
Voltage Source and Measure	200mV	±1e-10, ±180mV	smuX.SENSE_LOCAL
	200mV	±1e-10, ±180mV	smuX.SENSE_REMOTE
	2V	±1e-10, ±1.8V	smuX.SENSE_LOCAL
	2V	±1e-10, ±1.8V	smuX.SENSE_CALA
	20V	±1e-10, ±18V	smuX.SENSE_LOCAL
	200V	±1e-10, ±180V	smuX.SENSE_LOCAL
Current Source and Measure	100nA	±1e-10, ±90nA	smuX.SENSE_LOCAL
	1µA	±1e-10, ±0.9µA	smuX.SENSE_LOCAL
	10µA	±1e-10, ±9µA	smuX.SENSE_LOCAL
	100µA	±1e-10, ±90µA	smuX.SENSE_LOCAL
	1mA	±1e-10, ±0.9mA	smuX.SENSE_LOCAL
	1mA	±1e-10, ±0.9mA	smuX.SENSE_CALA
	10mA	±1e-10, ±9mA	smuX.SENSE_LOCAL
	100mA	±1e-10, ±90mA	smuX.SENSE_LOCAL
	1A	±1e-10, ±0.9A	smuX.SENSE_LOCAL
	1.5A	±1e-10, ±1.35A	smuX.SENSE_LOCAL
	10A	±1e-10, ±2.4A	smuX.SENSE_LOCAL

1. Calibrate only the source for the SENSE_CALA sense steps.
2. Steps must be performed in the order shown.
3. Do not use actual 0 values for zero calibration points. Send very small values such as ±1e-10. Calibration polarities must also be set as shown in the procedures.
4. Output must be off before changing to the CALA sense mode.

Table 16-4
Model 2635/2636 calibration steps

Function[1]	Calibration steps[2]	Calibration points[3]	Sense mode[4]
Voltage Source and Measure	200mV	±1e-12, ±180mV	smuX.SENSE_LOCAL
	200mV	±1e-12, ±180mV	smuX.SENSE_REMOTE
	2V	±1e-12, ±1.8V	smuX.SENSE_LOCAL
	2V	±1e-12, ±1.8V	smuX.SENSE_CALA
	20V	±1e-12, ±18V	smuX.SENSE_LOCAL
	200V	±1e-12, ±180V	smuX.SENSE_LOCAL
Current Source and Measure	10nA	±1e-12, +/-9nA	smuX.SENSE_LOCAL
	1nA	±1e-12, +/-0.9nA	smuX.SENSE_LOCAL
	100pA[5]	±1e-12, +/-90pA	smuX.SENSE_LOCAL
	100nA	±1e-12, ±90nA	smuX.SENSE_LOCAL
	1µA	±1e-12, ±0.9µA	smuX.SENSE_LOCAL
	10µA	±1e-12, ±9µA	smuX.SENSE_LOCAL
	100µA	±1e-12, ±90µA	smuX.SENSE_LOCAL
	1mA	±1e-12, ±0.9mA	smuX.SENSE_LOCAL
	1mA	±1e-12, ±0.9mA	smuX.SENSE_CALA
	10mA	±1e-12, ±9mA	smuX.SENSE_LOCAL
	100mA	±1e-12, ±90mA	smuX.SENSE_LOCAL
	1A	±1e-12, ±0.9A	smuX.SENSE_LOCAL
	1.5A	±1e-12, ±1.35A	smuX.SENSE_LOCAL

1. Calibrate only the source for the SENSE_CALA sense steps.
2. Steps must be performed in the order shown.
3. Do not use actual 0 values for zero calibration points. Send very small values such as ±1e-10. Calibration polarities must also be set as shown in the procedures.
4. Output must be off before changing to the CALA sense mode.
5. For Current Measure only.

Parameter values

The full-scale parameters are actually 90% of full scale as indicated in Table 16-2, Table 16-3, and Table 16-5. 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-10 or -1e-10.

Sense modes

Table 16-2, Table 16-3, and Table 16-5 list sense modes for the calibration steps. Note that all source and measure ranges are calibrated using the LOCAL sense mode. In addition, the 100mV or 200mV source and measure ranges are also calibrated using the REMOTE sense mode, and the 1mA and 1V or 2V source ranges are also calibrated using the CALA sense mode.

Calibration commands

Table 16-5 summarizes remote calibration commands. For a more complete description of these commands, refer to Section 12.

Table 16-5
Calibration commands

Command[1]	Description
smuX.cal.adjustdate = caldate	Set calibration adjustment date for 2635/2636 only.
smuX.cal.date = caldate	Set calibration date (caldate of 0 indicated date not set).
smuX.cal.due = caldue	Set calibration due date (caldue of 0 indicated date not set).
smuX.cal.lock()	Lock out calibration.
smuX.cal.password = "newpassword"	Change password to "newpassword".
smuX.cal.polarity = polarity	Set polarity: smuX.CAL_AUTO (auto polarity). smuX.CAL_NEGATIVE (negative polarity). smuX.CAL_POSITIVE (positive polarity).
smuX.cal.restore([calset])	Load calibration set of constants: smuX.CALSET_NOMINAL (nominal constants). smuX.CALSET_FACTORY (factory constants). smuX.CALSET_DEFAULT (normal constants). smuX.CALSET_PREVIOUS (previous constants).
smuX.cal.save()	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")	Unlock calibration (default password: KI0026XX)[2]
smuX.measure.calibratei(range, cp1measured, cp1reference, cp2measured, cp2reference)	Calibrate current measure range:[3] ±range (measurement range to calibrate). cp1measured (Series 2600 measured value for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (Series 2600 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:[2] ±range (measurement range to calibrate). cp1measured (Series 2600 measured value for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (Series 2600 measured value for cal. point 2). cp2reference (reference measurement for cal. point 2).

1. smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.
2. If you have version 1.1.4 firmware, or earlier, then your passwords are KI002601 for the Model 2601, and KI002602 for the Model 2602.
3. 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 or Table 16-3 for calibration points.

Table 16-5 (cont.)
Calibration commands

Command[1]	Description
smuX.source.calibratei(range, cp1expected, cp1reference, cp2expected, cp2reference)	Calibrate current source range:[2] ±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:[2] ±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.contact.calibratelo (cp1measured, cp1reference, cp2measured, cp2reference)	Calibrate the low/sense low contact check measurement. cp1measured (value measured by SMU for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (value measured by SMU for cal. point 2). cp2reference (reference measurement for cal. point 2).
smuX.contact.calibratehi (cp1measured, cp1reference, cp2measured, cp2reference)	Calibrate the high/sense high contact check measurement. cp1measured (value measured by SMU for cal. point 1). cp1reference (reference measurement for cal. point 1). cp2measured (value measured by SMU for cal. point 2). cp2reference (reference measurement for cal. point 2).

1. smuX = smua for the Model 2601/2611/2635; smuX = smua (Channel A) or smub (Channel B) for the Model 2602/2612/2636.
2. If you have version 1.1.4 firmware, or earlier, then your passwords are KI002601 for the Model 2601, and KI002602 for the Model 2602.
3. 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 or Table 16-3 for calibration points.

Calibration procedure

Step 1. Prepare the Series 2600 for calibration

a. Connect the Series 2600 to the controller IEEE-488 interface or RS-232 port using a shielded interface cable.
b. Turn on the Series 2600 and the test equipment, and allow them to warm up for at least two hours before performing calibration.
c. Make sure the IEEE-488 or RS-232 interface parameters are set up properly (press MENU > RS232 to configure the interface).

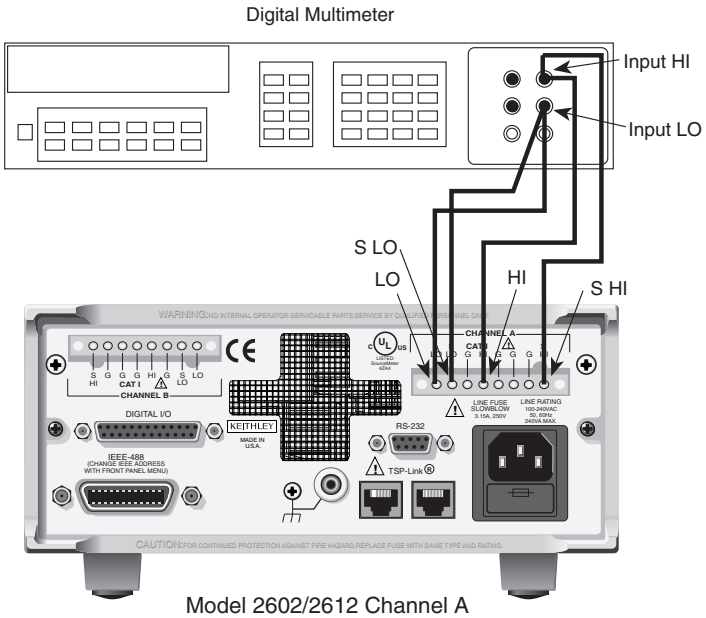
Step 2. Voltage Calibration

a. Connect the Series 2600 to the digital multimeter using the 4-wire connections shown in Figure 16-1, and select the multimeter DC volts function.
b. Send the following commands in order to initialize voltage calibration:

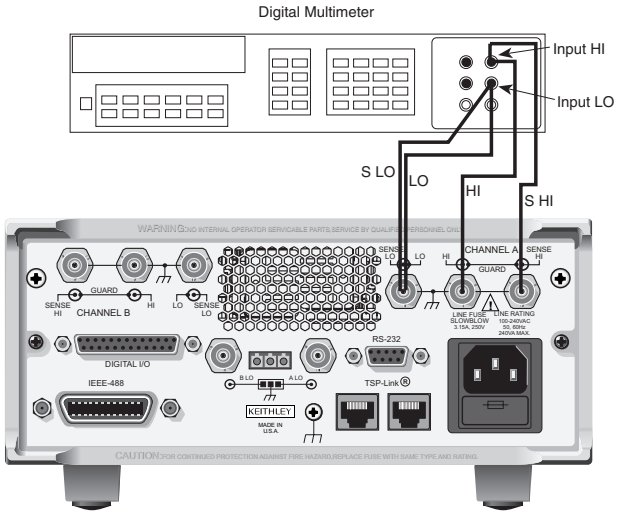
```
smua.cal.unlock("KI0026XX")
smua.reset()
smua.source.func = smua.OUTPUT_DCVOLTS
```

NOTE (For firmware versions earlier then 1.2.0, 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



Model 2602/2612 Channel A Connections Shown



Model 2636 channel A connections shown

c. Perform each calibration step listed in Table 16-2 (Model 2601/2601) ,or Table 16-3 (Model 2611/2612), or Table 16-4 (Model 2635/2636) as follows:

1) Select the range being calibrated with this command:

```
smua.source.rangev = range
```

NOTE (Note that it is not necessary to set the measure range for calibration.) For example, for the Model 2601/2602 1V range, the following command would be sent:

```
smua.source.rangev = 1
```

For the Model s2611/2612/2635/2636 2V range, the following command would be sent:

```
smua.source.rangev = 2
```

2) Select the correct sense mode based on the calibration step from Table 16-2, or Table 16-3, or Table 16-4, for example:

```
smua.sense = smua.SENSE_LOCAL
```

3) Select positive polarity, then set the source output to the positive zero value:

```
smua.cal.polarity = smua.CAL_POSITIVE
smua.source.levelv = 1e-10
```

4) Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

5) Allow the readings to settle, then get both the multimeter and Series 2600 voltage readings at the positive zero value (the Series 2600 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 Series 2600:

```
Z_rdg = smua.measure.v()
```

6) Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

7) Set the source output to the positive full scale value for the present range, for example:

```
smua.source.levelv = 0.9
```
Model 2601/2602

```
smua.source.levelv = 1.8
```
Model 2611/2612/2635/2636

8) Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

9) Allow the readings to settle, then get both the multimeter and Series 2600 voltage readings at the positive full-scale output value (the Series 2600 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 Series 2600:

```
FS_rdg = smua.measure.v()
```

10) Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

11) Send the source calibration command using the range, +zero and +FS multimeter readings, and +zero and +FS source values for the parameters:

```
smua.source.calibratev(range,src_Z,DMM_Z_rdg,
src_FS,DMM_FS_rdg)
```

Where:

`range`	present calibration range
`src_Z`	+zero 2600 source output value
`DMM_Z_rdg`	+zero DMM measurement
`src_FS`	+FS 2600 source output value
`DMM_FS_rdg`	+FS DMM measurement

Typical values for the Model 2601/2602 1V range:

```
smua.source.calibratev(1,1e-10,1e-5,0.9,0.903)
```

12) Typical values for the Models 2611/2612/2635/2636 2V range:

```
smua.source.calibratev(2,1e-10,1e-5,1.8,1.802)
```

13) If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and Series 2600 readings, and range setting for the parameters:

14) smua.measure.calibratev(range,Z_rdg,DMM_Z_rdg, FS_rdg,DMM_FS_rdg)

Where:

`range`	present calibration range
`Z_rdg`	+zero 2600 measurement
`DMM_Z_rdg`	+zero DMM measurement
`FS_rdg`	+FS 2600 measurement
`DMM_FS_rdg`	+FS DMM measurement

Typical Model 2601/2602 1V range values:

```
smua.measure.calibratev(1,1e-4,1e-5,0.92,0.903)
```

Typical Model 2611/2612/2635/2636 2V range values:

```
smua.measure.calibratev(2,1e-4,1e-5,1.82,1.802)
```

15) 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-10
```

16) Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

17) Allow the readings to settle, then get both the multimeter and Series 2600 voltage readings at the negative zero value. (The Series 2600 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 Series 2600:

```
Z_rdg = smua.measure.v()
```

18) Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

19) Set the source output to the negative full scale value, for example:

`smua.source.levelv = -0.9`(Model 2601/2602)
`smua.source.levelv = -1.8`(Model 2611/2612/2635/2636)

20) Turn on the output:

```
smua.source.output = smua.OUTPUT_ON
```

21) Allow the readings to settle, then get both the multimeter and Series 2600 voltage readings at the negative full-scale output value (the Series 2600 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 Series 2600:

```
FS_rdg = smua.measure.v()
```

22) Turn off the output:

```
smua.source.output = smua.OUTPUT_OFF
```

23) 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 2600 source output value
`DMM_Z_rdg`	-zero DMM measurement
`src_FS`	-FS 2600 source output value
`DMM_FS_rdg`	-FS DMM measurement

Typical values for the Model 2601/2602 1V range:

```
smua.source.calibratev(-1,-1e-10,-1e-4,-0.9,-0.896)
```

24) Typical values for the Model 2611/2612/2635/2636 2V range:
```
smua.source.calibratev(-2,-1e-10,-1e-4,-1.8,-1.805)
```
25) If this step is not on the CALA sense mode, send the measure calibration command using the multimeter and Series 2600 readings, and range setting for the parameters:
26) smua.measure.calibratev(-range,Z_rdg,DMM_Z_rdg, FS_rdg,DMM_FS_rdg)

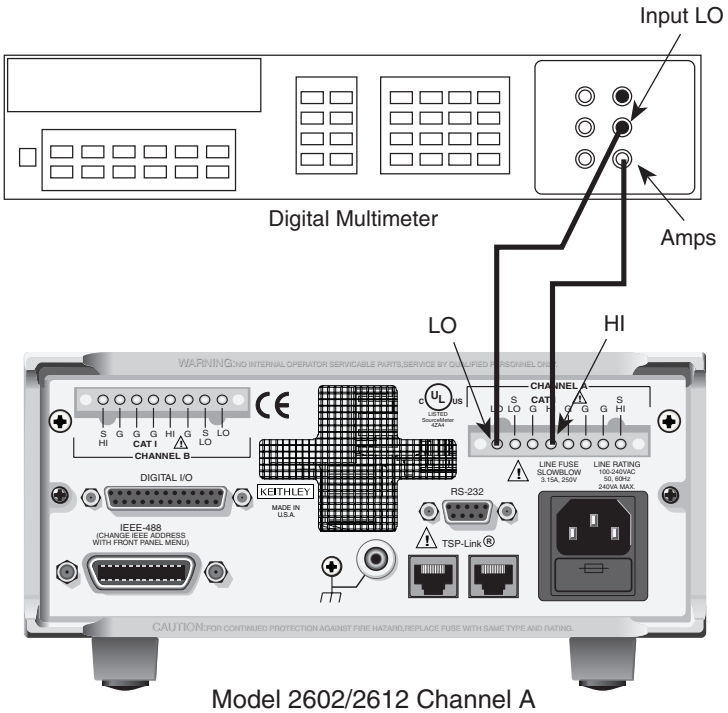
Where:		
	`-range`	negative of the present calibration range
	`Z_rdg`	-zero 2600 measurement
	`DMM_Z_rdg`	-zero DMM measurement
	`FS_rdg`	-FS 2600 measurement
	`DMM_FS_rdg`	-FS DMM measurement

Typical Model 2601/2602 1V range values:
```
smua.measure.calibratev(-1,-1e-4,-1e-6,-0.89,-0.896)
```
27) Typical Model 2611/2612/2635/2636 2V range values:
```
smua.measure.calibratev(-2,-1e-4,-1e-6,-1.81,-1.805)
```

d. Be sure to complete steps a through v for all six voltage steps in Table 16-2, Table 16-3, or Table 16-5 before continuing with current calibration.

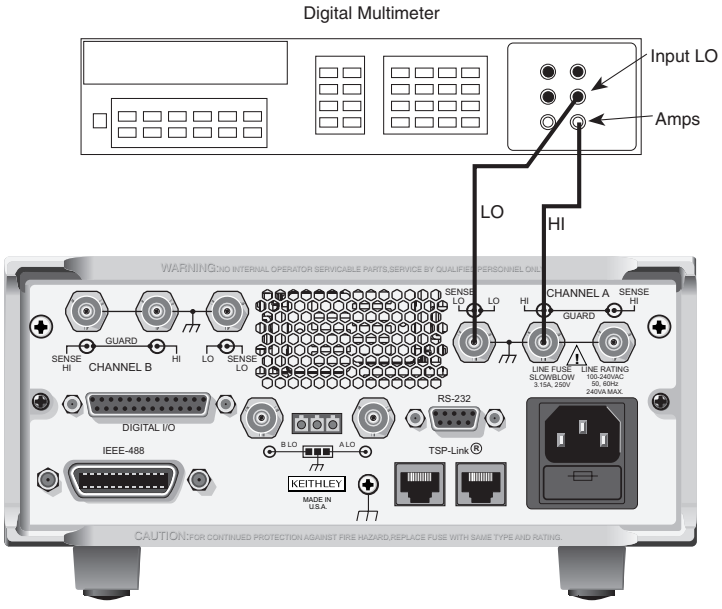
e. Select auto polarity mode:
```
smua.cal.polarity = smua.CAL_AUTO
```

Figure 16-2
Connections for current calibration (100nA to 1A ranges)



Model 2602/2612 Channel A
Connections Shown

Connections for current calibration (1uA to 1A ranges)



Model 2636 channel A connections shown

Step 3. Current Calibration

Models 2601/2602/2611/2612:

1. Connect the Series 2600 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 (Model 2601/2602), Table 16-3 (Model 2611/2612), or Table 16-4 (Model 2635/2636):
   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, Table 16-3, or Table 16-4, 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.leveli = 1e-10
      ```
   d. Turn on the output:
      ```
      smua.source.output = smua.OUTPUT_ON
      ```
   e. Allow the readings to settle, then get both the multimeter and Series 2600 current readings at the positive zero value (the Series 2600 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 Series 2600:
      ```
      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.leveli = 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 Series 2600 voltage readings at the positive full-scale output value (the Series 2600 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 Series 2600:
      ```
      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 2600 source output value
`DMM_Z_rdg`	+zero DMM measurement
`src_FS`	+FS 2600 source output value
`DMM_FS_rdg`	+FS DMM measurement

Typical values for the 1A range:
```
smua.source.calibratei(1,1e-10,1e-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 Series 2600 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 2600 measurement
`DMM_Z_rdg`	+zero DMM measurement
`FS_rdg`	+FS 2600 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.leveli = -1e-10
```

n. Turn on the output:
```
smua.source.output = smua.OUTPUT_ON
```

o. Allow the readings to settle, then get both the multimeter and Series 2600 current readings at the negative zero full-scale value (the Series 2600 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 Series 2600:
```
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.leveli = -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 Series 2600 current readings at the negative full-scale output value (the Series 2600 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 Series 2600:
```
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 2600 source output value
`DMM_Z_rdg` -zero DMM measurement
`src_FS` -FS 2600 source output value
`DMM_FS_rdg` -FS DMM measurement

Typical values for the 1A range:

```
smua.source.calibratei(-1,-1e-10,-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 Series 2600 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 2600 measurement
`DMM_Z_rdg` -zero DMM measurement
`FS_rdg` -FS 2600 measurement
`DMM_FS_rdg` -FS DMM measurement

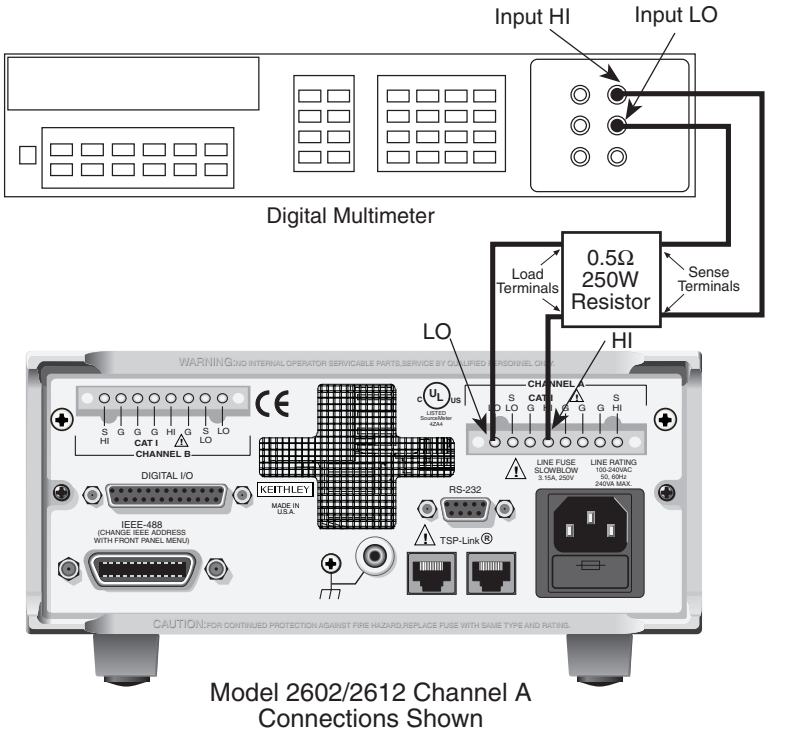
Typical 1A range values:

```
smua.measure.calibratei(-1,-1e-4,-1e-5,-0.91,-0.892)
```

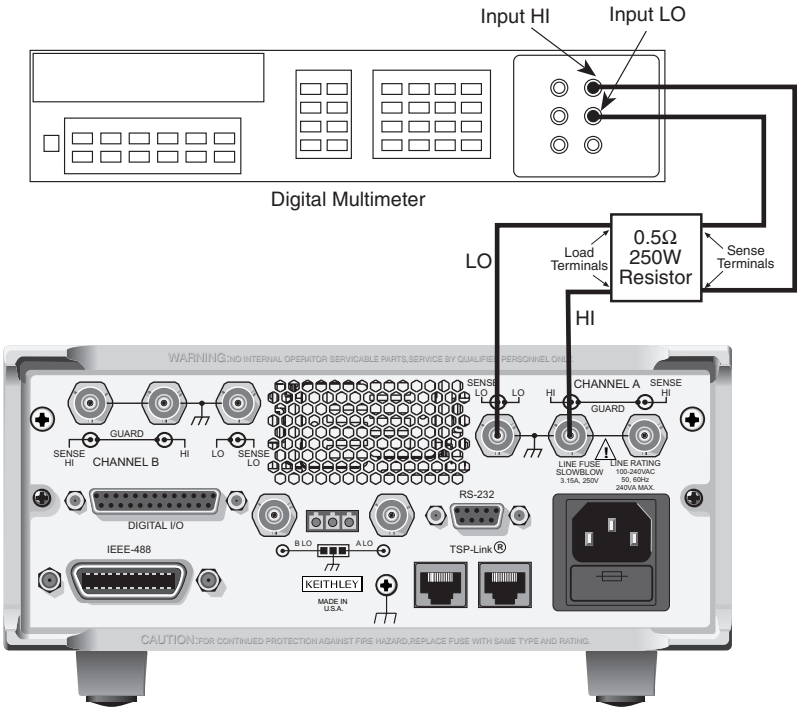
4. Be sure to complete steps a through v for the 100nA to 1A ranges before continuing with 3A and 10A range calibration (Model 2601/2602) or 1.5A and 10A range calibration (Model 2611/2612/2635/2636).
5. Change connections as shown in Figure 16-3 (use 4-wire connections to the 0.5Ω resistor as shown).
6. Select the DMM DC volts function.
7. Repeat steps a through v for the 3A and 10A ranges (Model 2601/2602) or 1.5A and 10A ranges (Model 2611/2612/2635/2636). 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



Model 2602/2612 Channel A Connections Shown



Model 2636 channel A connections shown

Models 2635 and 2636:

1. Connect the Series 2600 to the digital multimeter (see Figure 16-2), and select the multimeter DC current function.
2. Calibrate the low current ranges (100pA, 1nA, 10nA, 100nA[1]) using a suitably guarded and characterized 1GΩ resistance standard, such as the Keithley Instruments Model 2600-STD-RES. (see Table 16-1, "Recommended calibration equipment").Step-by-step procedures, connection diagrams, and a factory script for calibrating the low current ranges are included with the Model 2600-STD-RES. The general process entails forcing a characterized voltage across the 1GΩ resistor and comparing the 2635/2636 measured results against the standard resistance and voltage derived current.
   a. Characterize the appropriate +/- V source values with the Digital Multimeter according to Table 16-4.
   b. Characterize the desired Model 2635/2636 current ranges.
      1) Connect the guarded resistance standard.
      2) Source the appropriate voltage for +/- full scale reading.
      3) Wait 30 seconds for stable measurement.
      4) Capture the Model 2635/2636 reported current measurement.
      5) Initiate HI-Z mode to open the resistor standard (source zero current) and the characterize offset.
      6) Repeat the above steps for each low current range.

Table 16-6
Settings of Model 2635/2636 Characterization of Voltage Source

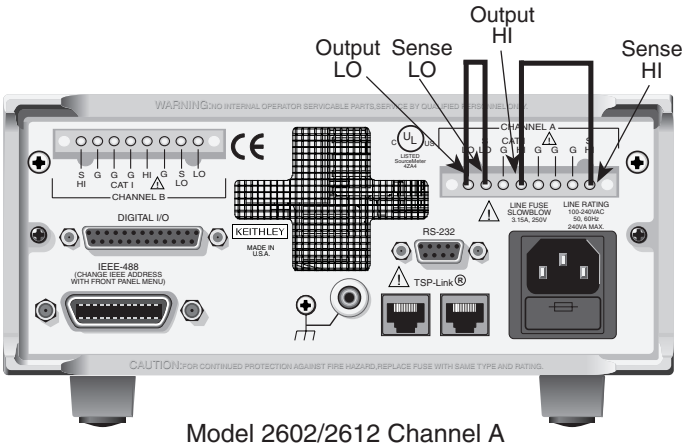
Low Current Range	Voltage Source	Compliance
100pA	+/- 100.00mV	1.5A
1nA	+/- 1.0000mV	1.5A
10nA	+/- 10.000mV	1.5A
100nA	+/- 100.00V	100mA

Step 4. Contact check calibration (For 2601/2602/2611/2612 only)

a. Short the Series 2600 sense low and output low terminals, as shown in Figure 16-4. Also short the sense high and output high terminals together, as shown in the figure.

1.The 2601/02/11/12 could be calibrated with this method for the 100nA setting only if desired.

Figure 16-4
Connections for contact check 0Ω calibration



Model 2602/2612 Channel A connections shown

b. Allow the readings to settle, then get the Series 2600 readings:
```
r0_hi, r0_lo = smua.contact.r()
```

c. Connect a 50Ω resistor between the sense low and output low terminals, as shown in Figure 16-5. Also connect a second 50Ω resistor between the sense high and output high terminals as shown in the figure.

d. Allow the readings to settle, then get the Series 2600 readings:
```
r50_hi, r50_lo = smua.contact.r()
```

e. Send the contact check low calibration command:
```
smua.contact.calibratelo(r0_lo, Z_actual,r50_lo,50_ohm_actual)
```
Where:

`r0_lo`	Series 2600 0Ω low measurement
`Z_actual`	actual zero value
`r50_lo`	Series 2600 50Ω low measurement
`50_ohm_actual`	actual 50Ω resistor value

Typical values:
```
smua.contact.calibratelo(r0_lo,0,r50_lo,50.15)
```

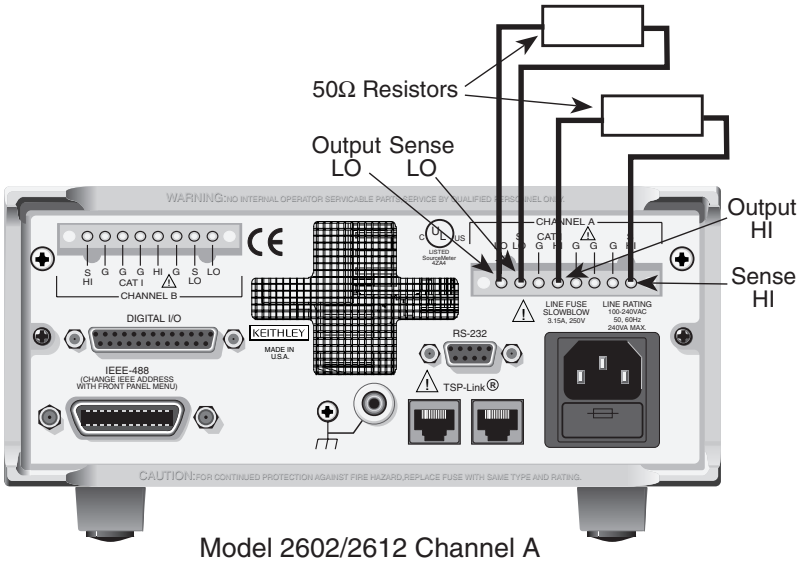
f. (Where `r0_lo` and `r50_lo` are measurements taken in steps 2 and 4 above.)

g. Send the contact check high calibration command:
```
smua.contact.calibratehi(r0_hi, Z_actual,r50_hi,50_ohm_actual)
```
Where:

`r0_hi`	Series 2600 0Ω high measurement
`Z_actual`	actual zero value
`r50_hi`	Series 2600 50Ω high measurement
`50_ohm_actual`	actual 50Ω resistor value

Typical values:
```
smua.contact.calibratehi(r0_hi,0,r50_hi,50.15)
```
(Where `r0_hi` and `r50_hi` are measurements taken in steps 2 and 4 above.)

Figure 16-5
Connections for contact check 50Ω calibration



Model 2602/2612 Channel A
connections shown

Step 5. Program calibration dates

Use the following command to set the calibration adjustment date (this is Model 2635 and 2636 only):

```
smua.cal.adjustdate = os.time{year=2005, month=1, day=1}
```

Optionally, it is possible to set the calibration date and calibration due date with the following commands (this is required by models 2601, 2602, 2611 and 2612):

```
smua.cal.date = os.time{year=2005, month=1, day=1}
smua.cal.due = os.time{year=2006, month=1, day=1}
```

If you do not wish to set a calibration date or calibration due date, you can use the following commands (this is model 2635 and 2636 only):

```
smua.cal.date = 0
smua.cal.due = 0
```

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 6. 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 7. Lock out calibration

To lock out further calibration, send the following command after completing the calibration procedure:

```
smua.cal.lock()
```

Step 8. Repeat calibration procedure for Model 2602/2612/2636 Channel B

For the Models 2602, 2612, and 2636 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.

This page left blank intentionally.

Section 17

Routine Maintenance

In this section:

Topic	Page
Introduction	**17-2**
Line fuse replacement	**17-2**
Front panel tests	**17-3**
Keys test	17-3
Display Patterns test	17-3

Introduction

The information in this section deals with routine maintenance of the Keithley Instruments Series 2600 System SourceMeter® that can be performed by the operator.

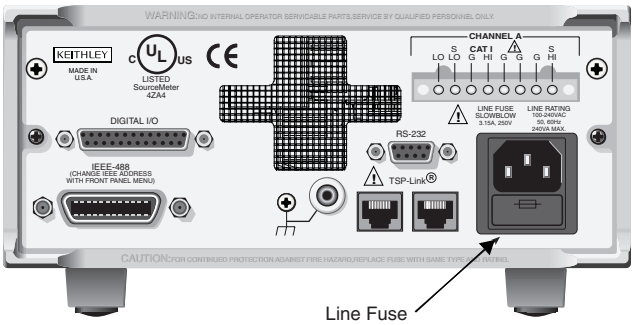
Line fuse replacement

WARNING ***Disconnect the line cord at the rear panel, and remove all test leads connected to the instrument before replacing the line fuse. Failure to do so could expose the operator to hazardous voltages that could result in personal injury or death.***

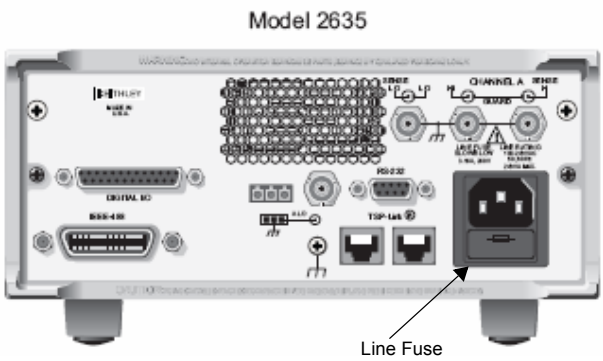
NOTE The power line fuse is accessible from the rear panel, just below the AC power receptacle (Figure 17-1).

Figure 17-1
Line fuse replacement

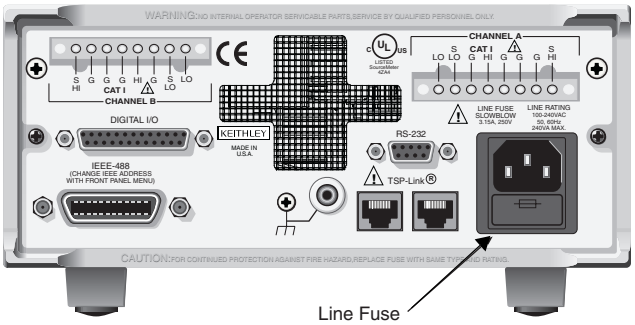
Model 2601/2611



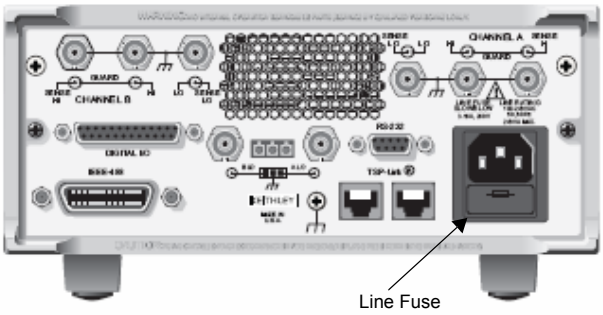
Model 2635



Model 2602/2612



Model 2636



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 Instruments 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. Press MENU > Display > TEST.
2. Select DISPLAY-TESTS, and press ENTER or the navigation wheel
3. Choose one of the folowing:
   - KEYS
   - DISPLAY_PATTERNS
4. Select KEYS, and press ENTER or the navigation wheel 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 DISPLAY 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. Press MENU > DISPLAY > TEST.
2. Select DISPLAY-TEST, and press ENTER or the navigation wheel to display the following menu:

3. Choose one of the following:
   - FRONT PANEL TESTS
   - KEYS DISPLAY_PATTERNS
4. Select DISPLAY_PATTERNS, and press ENTER or the Navigation Wheel to start the display test. There are three parts to the display test. Each time ENTER or Navigation Wheel is pressed, the next part of the test sequence is selected. The three parts of the test sequence are as follows:
   - Checkerboard pattern and the 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.

Appendix A
Specifications

In this appendix:

Topic	Page
Models 2601/2602 System SourceMeter® Specifications	**A-2**
Models 2611/2612 System SourceMeter® Specifications	A-9
Model 2635/2636 System SourceMeter® Specifications	A-15
Series 2600 System SourceMeter® Specifications	A-23

KEITHLEY

Models 2601/2602

System SourceMeter® Specifications

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

1. SPECIFICATION CONDITIONS

This document contains specifications and supplemental information for the Keithley Instruments Models 2601 and 2602 System SourceMeters®. Specifications are the standards against which the Models 2601 and 2602 are tested. Upon leaving the factory, the Models 2601 and 2602 meet these specifications. Supplemental and typical values are nonwarranted, apply at 23°C, and are provided solely as useful information.

The source and measurement accuracies are specified at the SourceMeter CHANNEL A (2601 and 2602) or SourceMeter CHANNEL B (2602) terminals under the following conditions:

1. 23°C ± 5°C, <70% relative humidity.
2. After two-hour warm-up.
3. Speed normal (1 NPLC).
4. A/D auto-zero enabled.
5. Remote sense operation or properly zeroed local operation.
6. Calibration period: one year.

2. SOURCE SPECIFICATIONS

Voltage Programming Accuracy[1]

Range	Programming resolution	Accuracy (1 year) 23°C ± 5°C ± (% rdg.+ volts)	Typical noise (peak-peak) 0.1Hz-10Hz
100.000mV	5µV	0.02% + 250µV	20µV
1.00000V	50µV	0.02% + 400µV	50µV
6.00000V	50µV	0.02% + 1.8mV	100µV
40.0000V	500µV	0.02% + 12mV	500µV

Temperature coefficient (0°C–18°C and 28°C–50°C): ±(0.15 × accuracy specification)/°C.

Maximum output power and source/sink limits:[2] 40.4W per channel maximum. ±40.4V at ±1.0A, ±6.06V at ±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:[3] 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).

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 (quadrants 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.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Current Programming Accuracy

Range	Programming resolution	Accuracy (1 year) 23°C ± 5°C ± (% rdg.+amps)	Typical noise (peak-peak) 0.1Hz-10Hz
100.000nA	1pA	0.06% + 100pA	5pA
1.00000µA	10pA	0.03% + 800pA	25pA
10.0000µA	100pA	0.03% + 5nA	60pA
100.000µA	1nA	0.03% + 60nA	3nA
1.00000mA	10nA	0.03% + 300nA	6nA
10.0000mA	100nA	0.03% + 6µA	200nA
100.000mA	1µA	0.03% + 30µA	600nA
1.00000A	10µA	0.05% + 1.8mA	70µA
3.00000A	10µA	0.06% + 4.0mA	150µA

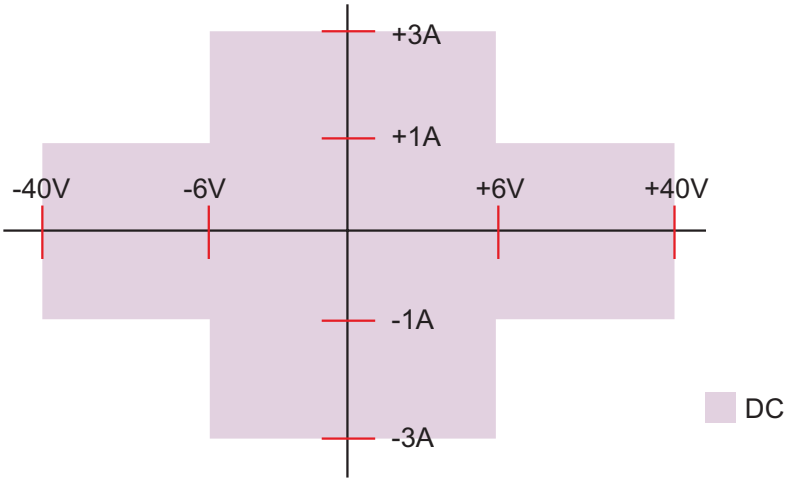
Temperature coefficient (0°C–18°C and 28°C–50°C): ±(0.15 × accuracy specification)/°C.

Maximum output power and source/sink limits:[2] 40.4W per channel maximum. ±1.01A at ±40.0V, ±3.03A at ±6.0V, four quadrant source or sink operation.

Current regulation: Line: 0.01% of range. **Load:** ±(0.01% of range + 100pA).

Voltage limit/compliance:[4] 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).



Models 2601 and 2602 I-V capability

4 For sink mode operation (quadrants 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.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

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, when changing from 10% to 90% of range, after source level command is processed on a fixed range.

100mV, 1V ranges: <50µs typical.
6V Range: <100µs typical.
40V Range: <150µs typical.[5]

Current source output settling time: Time required to reach 0.1% of final value, when changing from 10% to 90% of range, after source level command is processed on a fixed range. Values below for Iout · Rload = 1V unless noted.

3A–10mA ranges: <80µs typical (current less than 2.5A, Rload >1.5Ω).

1mA range: <100µs typical.

100µA range: <150µs typical.

10µA range: <500µs typical.

1µA range: <2.5ms typical.

100nA range: <25ms typical.

DC floating voltage: Output can be floated up to ±250VDC from chassis ground.

Remote sense operating range:[1]
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% of larger range + 300mV/Rload typical.
(see “CURRENT SOURCE OUTPUT SETTLING TIME” for additional test conditions).

5 Add 150µs when measuring on the 1A range.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

3. METER SPECIFICATIONS

Voltage Measurement Accuracy[6]

Range	Display resolution[7]	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°C–18°C and 28°C–50°C): ±(0.15 × accuracy specification)/°C.

Current Measurment Accuracy

Range	Display resolution[7]	Voltage burden[8]	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% + 1.5nA
100.000µA	1nA	<1mV	0.02% + 25nA
1.00000mA	10nA	<1mV	0.02% + 200nA
10.0000mA	100nA	<1mV	0.02% + 2.5µA
100.000mA	1µA	<1mV	0.02% + 20µA
1.00000A	10µA	<1mV	0.03% + 1.5mA
3.00000A	10µA	<1mV	0.05% + 3.5mA

Temperature coefficient (0°C–18°C and 28°C–50°C): ±(0.15 × accuracy specification)/°C.

6 Add 50µV to source accuracy specifications per volt of HI lead drop.
7 Applies when in single channel display mode.
8 Four-wire remote sense only.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Contact Check[9]

Speed	Maximum measurement time to memory for 60Hz (50Hz)[9]	Accuracy (1 year) 23°C ± 5°C ± (% rdg.+ ohms)
Fast	1 (1.2)ms	5% + 10
Medium	4 (5)ms	5% + 1
Slow	36 (42)ms	5% + 0.3

Additional Meter Specifications

Load impedance: Stable into 10,000pF typical.
Common mode voltage: 250VDC.
Common mode isolation: >1GΩ, <4500pF.
Over-range: 101% of source range, 102% of measure range.
Maximum sense lead resistance: 1kΩ for rated accuracy.
Sense input impedance: >10GΩ.

9 Includes measurement of SENSE HI to HI and SENSE LO to LO contact resistances.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

4. GENERAL

Host Interfaces: Computer control interfaces.

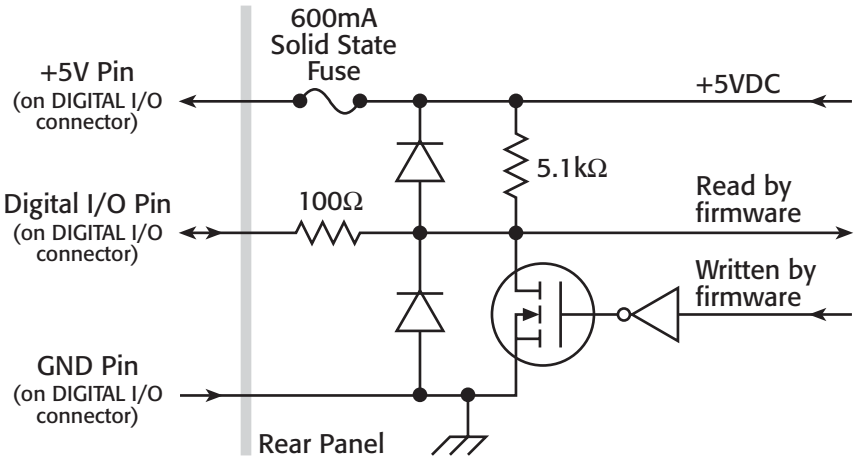
IEEE-488: IEEE-488.1 compliant. Supports IEEE-488.2 common commands and status model topology.

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 open drain I/O bits.
Absolute maximum input voltage: 5.25V.
Absolute minimum input voltage: –0.25V.
Maximum logic low input voltage: 0.7V, +850µA max.
Minimum logic high input voltage: 2.1V, +570µA.
Maximum source current (flowing out of digital I/O bit): +960µA.
Maximum sink current at maximum logic low voltage (0.7V): –5.0mA.
Absolute maximum sink current (flowing into digital I/O pin): –11mA.
5V power supply pin: Limited to 600mA, solid state fuse protected.



Output enable pin: Active high input pulled down internally to ground with 10kΩ resistor. When the Output Enable input function has been activated, each SourceMeter channel will not turn on unless the output enable pin is driven to >2.1V (nominal current = 2.1V / 10kΩ = 210µA).

Power Supply: 100V to 250VAC, 50Hz–60Hz (auto sensing), 250VA max.

Cooling: Forced air. Side intake and rear exhaust. One side must be unobstructed when rack-mounted.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

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 × 213mm wide × 460mm deep (3 1⁄2 in × 8 3⁄8 in × 17 1⁄2 in). Bench configuration (with handle & feet): 104mm high × 238mm wide × 460mm deep (4 1⁄8 in × 9 3⁄8 in × 17 1⁄2 in).

Weight: 2601: 4.75kg (10.4 lbs). **2602:** 5.50kg (12.0 lbs).

Environment: For indoor use only.

Altitude: Maximum 2000 meters above sea level.
Operating: 0°C–50°C, 70% R.H. up to 35°C. Derate 3% R.H./°C, 35°C–50°C.
Storage: –25°C to 65°C.

KEITHLEY

Models 2611/2612

System SourceMeter® Specifications

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

1. SPECIFICATION CONDITIONS

This document contains specifications and supplemental information for the Keithley Instruments Models 2611 and 2612 System SourceMeters®. Specifications are the standards against which the Models 2611 and 2612 are tested. Upon leaving the factory, the Models 2611 and 2612 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 (Models 2611 and 2612) or SourceMeter CHANNEL B (Model 2612) terminals under the following conditions:

1. 23°C ± 5°C, <70% relative humidity.
2. After two-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: one year.

2. SOURCE SPECIFICATIONS

Voltage Programming Accuracy[1]

Range	Programming resolution	Accuracy (1 year) 23°C ± 5°C ± (% rdg.+ volts)	Typical noise (peak-peak) 0.1Hz-10Hz
200.000mV	5µV	0.02% + 375µV	20µV
2.00000V	50µV	0.02% + 600µV	50µV
20.0000V	500µV	0.02% + 5mV	300µV
200.000V	5mV	0.02% + 50mV	2mV

Temperature coefficient (0°C–18°C and 28°C–50°C): ±(0.15 × accuracy specification)/°C.

Maximum output power and source/sink limits:[2] 30.603W per channel maximum. ±20.2V at ±1.515A, ±202V at ±101mA, four quadrant source or sink operation.

Voltage regulation: Line: 0.01% of range. **Load:** ±(0.01% of range + 100µV).

Noise 10Hz–20MHz: <5mV RMS typical, 20V range, 1A limit.

Current limit/compliance:[3] 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: <4mV (current ≤10mA).

---

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 (quadrants 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.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Current Programming Accuracy[4]

Range	Programming resolution	Accuracy (1 year) 23°C ± 5°C ± (% rdg.+amps)	Typical noise (peak-peak) 0.1Hz-10Hz
100.000nA	2pA	0.06% + 100pA	5pA
1.00000µA	20pA	0.03% + 800pA	25pA
10.0000µA	200pA	0.03% + 5nA	60pA
100.000µA	2nA	0.03% + 60nA	3nA
1.00000mA	20nA	0.03% + 300nA	6nA
10.0000mA	200nA	0.03% + 6µA	200nA
100.000mA	2µA	0.03% + 30µA	600nA
1.00000A[2]	20µA	0.05% + 1.8mA	70µA
1.50000A[2]	50µA	0.06% + 4mA	150µA
10.0000A[2,5]	200µA	0.5% + 40mA	

Temperature coefficient (0°C–18°C and 28°C–50°C): ±(0.15 × accuracy specification)/°C.

Maximum output power and source/sink limits:[2] 30.603W per channel maximum. ±1.515A at ±20.2V, ±101mA at ±202V, four-quadrant source or sink operation.

Current regulation: Line: 0.01% of range. **Load:** ±(0.01% of range + 100pA).

Voltage limit/compliance:[6] 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.

200mV, 2V ranges: <50µs typical. **20V range:** <110µs typical. **200V range:** <700µs 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.

1.5A–1A ranges: <120µs typical (Rload >6Ω).
100mA–10mA ranges: <80µs typical.
1mA range: <100µs typical.
100µA range: <150µs typical.

4 Accuracy specifications do not include connector leakage. Derate accuracy by Vout/2E11 per °C when operating between 18°C–28°C. Derate accuracy by Vout/2E11 + (0.15 * Vout/2E11) per °C when operating <18°C and >28°C.
5 10A range accessible only in pulse mode.
6 For sink mode operation (quadrants II and IV), add 10% of compliance range and ±0.02% of limit setting to corresponding voltage source specification. For 200mV range add an additional 120mV of uncertainty.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

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:[1] Maximum voltage between HI and SENSE HI = 3V. Maximum voltage between LO and SENSE LO = 3V.

Voltage output headroom:

200V range: Max. output voltage = 202.3V; total voltage drop across source leads (maximum 1Ω per source lead).
20V range: Max. output voltage = 23.3V; total voltage drop across source leads (maximum 1Ω per source lead).

Over temperature protection: Internally-sensed temperature overload puts unit in standby mode.

Voltage source range change overshoot: Overshoot into a 200k load, 20MHz BW, 300mV typical.

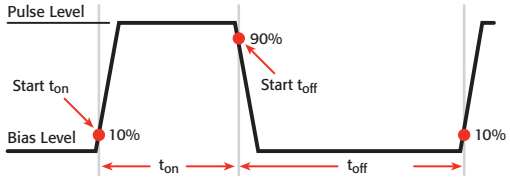
Current source range change overshoot: <5% of larger range + 300mV/Rload + 60nA typical (see "Current source output settling time" for additional test conditions).

Pulse Specifications

Region	Maximum current limit	Maximum pulse width[7]	Maximum duty cycle[8]
1	100mA at 200V	DC, no limit	100%
1	1.5A at 20V	DC, no limit	100%
2	1A at 180V	8.5ms	1%
3[9]	1A at 200V	2.2ms	1%
4	10A at 5V	1ms	2.2%

Minimum programmable pulse width:[7] 200µs. NOTE: Minimum pulse width for settled source at a given I/V output and load can be longer than 200µs. See note 10 for typical settling times.[10]

7 Times measured from the start of pulse to the start of off-time:



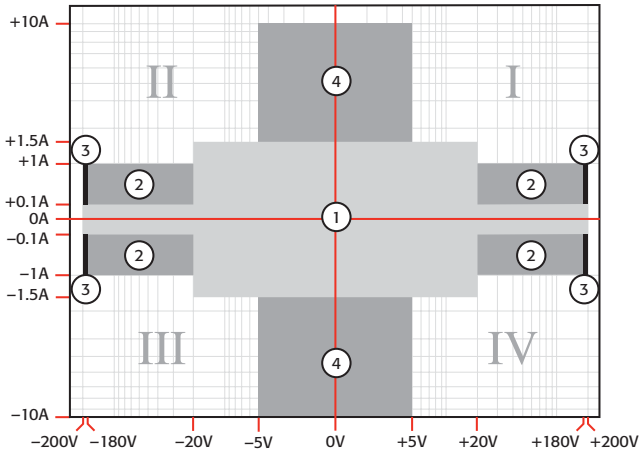
8 Thermally limited in sink mode (quadrants 2 and 4) and ambient temperatures above 30°C. See power equations in the Reference Manual for more information.
9 Voltage source operation with 1.5A current limit.
10 Typical performance for minimum settled pulse widths:

Source value	Load	Source settling (% of range)	Min. pulse width
5V	0.5Ω	1%	300µs
20V	200Ω	0.2%	200µs
180V	180Ω	0.2%	5ms
200V (1.5A limit)	200Ω	0.2%	1.5ms
100 mA	200Ω	1%	200µs
1A	20Ω	1%	500µs
1A	180Ω	0.2%	5ms
10A	0.5Ω	0.5%	300µs

Typical tests were performed using remote operation, 4W sense, Keithley 2600 ban cables and best fixed measurement range. For more information on pulse scripts, see the Series 2600 Reference Manual.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Pulse width programming resolution: 1µs.
Pulse width programming accuracy:[7] ±25µs.
Typical pulse width jitter: 50µs.



3. METER SPECIFICATIONS

Voltage Measurement Accuracy[11, 12]

Range	Display resolution[13]	Input resistance	Accuracy (1 year) 23°C ± 5°C ± (% rdg.+ volts)
200.000mV	1µV	>10GΩ	0.015% + 225µV
2.00000V	10µV	>10GΩ	0.02% + 350µV
20.0000V	100µV	>10GΩ	0.015% + 5mV
200.000V	1mV	>10GΩ	0.015% + 50mV

Temperature coefficient (0°C–18°C and 28°C–50°C): ±(0.15 × accuracy specification)/°C.

11 Add 50µV to source accuracy specifications per volt of HI lead drop.
12 De-rate accuracy specifications for NPLC setting <1 by increasing error term. Add appropriate % of range term using table below:

NPLC setting	200mV range	2V–200V ranges	100nA range	1µA–100mA ranges	1A–1.5A ranges
0.1	0.01%	0.01%	0.01%	0.01%	0.01%
0.01	0.08%	0.07%	0.1 %	0.05%	0.05%
0.001	0.8 %	0.6 %	1 %	0.5 %	1.1 %

13 Applies when in single channel display mode.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Current measurement accuracy[14,12]

Range	Display resolution[13]	Voltage burden[15]	Accuracy (1 year) 23°C ± 5°C ± (% rdg.+amps)
100.000nA	1pA	<1mV	0.05% + 100pA
1.00000µA	10pA	<1mV	0.025% + 500pA
10.0000µA	100pA	<1mV	0.025% + 1.5nA
100.000µA	1nA	<1mV	0.02% + 25nA
1.00000mA	10nA	<1mV	0.02% + 200nA
10.0000mA	100nA	<1mV	0.02% + 2.5µA
100.000mA	1µA	<1mV	0.02% + 20µA
1.00000A	10µA	<1mV	0.03% + 1.5mA
1.50000A	10µA	<1mV	0.05% + 3.5mA
10.0000A[16]	100µA	<1mV	0.4% + 25mA

Temperature Coefficient (0°C–18°C and 28°C–50°C): ±(0.15 × accuracy specification)/°C.

Contact Check[17]

Speed	Maximum measurement time to memory for 60Hz (50Hz)[17]	Accuracy (1 year) 23°C ± 5°C ± (% rdg.+ohms)
Fast	1 (1.2)ms	5% + 10
Medium	4 (5)ms	5% + 1
Slow	36 (42)ms	5% + 0.3

Additional Meter Specifications

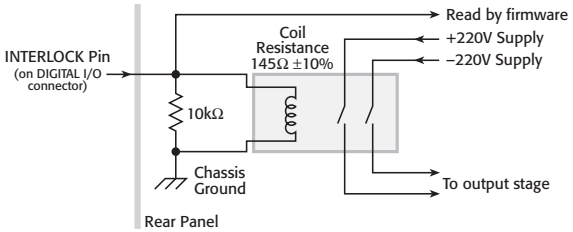
Load impedance: Stable into 10,000pF typical.
Common mode voltage: 250VDC.
Common mode isolation: >1GΩ, <4500pF.
Over-range: 101% of source range, 102% of measure range.
Maximum sense lead resistance: 1kΩ for rated accuracy.
Sense input impedance: >10GΩ.

14 De-rate accuracy by Vout/2E11 per °C when operating between 18°C–28°C. Derate accuracy by Vout/2E11 + (0.15 * Vout/2E11) per °C when operating <18°C and >28°C.
15 Four-wire remote sense only.
16 10A range accessible only in pulse mode.
17 Includes measurement of SENSE HI to HI and SENSE LO to LO contact resistances.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

4. 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 (see Models 2601 and 2602 GENERAL specifications for circuit diagram):
Connector: 25-pin female D.
Input/output pins: 14 open drain I/O bits.
Absolute maximum input voltage: 5.25V.
Absolute minimum input voltage: –0.25V.
Maximum logic low input voltage: 0.7V, +850µA max.
Minimum logic high input voltage: 2.1V, +570µA.
Maximum source current (flowing out of digital I/O bit): +960µA.
Maximum sink current at maximum logic low voltage (0.7V): –5.0mA.
Absolute maximum sink current (flowing into digital I/O pin): –11mA.
5V Power supply pin: Limited to 600mA, solid state fuse protected.
Safety interlock pin: Active high input. >3.4V at 24mA (absolute maximum of 6V) must be externally applied to this pin to ensure 200V operation. This signal is pulled down to chassis ground with a 10kΩ resistor. 200V operation will be blocked when the INTERLOCK signal is <0.4V (absolute minimum of –0.4V). See figure below:



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: One 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 × 213mm wide × 460mm deep (3 1⁄2 in × 8 3⁄8 in × 17 1⁄2 in). Bench configuration (with handle and feet): 104mm high × 238mm wide × 460mm deep (4 1⁄8 in × 9 3⁄8 in × 17 1⁄2 in).
Weight: Model 2611: 4.75kg (10.4 lbs). **Model 2612:** 5.50kg (12.0 lbs).
Environment: For indoor use only.
Altitude: Maximum 2000 meters above sea level.
Operating: 0°C–50°C, 70% R.H. up to 35°C. Derate 3% R.H./°C, 35°C–50°C.
Storage: –25°C to 65°C.

KEITHLEY

Model 2635/2636

System SourceMeter® Specifications

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

1. SPECIFICATION CONDITIONS

This document contains specifications and supplemental information for the Models 2635 and 2636 System SourceMeters®. Specifications are the standards against which the Models 2635 and 2636 are tested. Upon leaving the factory the 2635 and 2636 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 SourceMeters® CHANNEL A (2635 and 2636) or SourceMeters® B (2636) terminals under the following conditions:

1. 23°C ± 5°C, < 70% relative humidity.
2. After two-hour warm-up.
3. Speed normal (1 NPLC).
4. A/D auto-zero enabled.
5. Remote sense operation or properly zeroed local operation.
6. Calibration period: one year.

2. SOURCE SPECIFICATIONS

VOLTAGE SOURCE SPECIFICATIONS

Specifications Category	Specifications			
Voltage Programming Accuracy[1]	**RANGE**	**PROGRAMMING RESOLUTION**	**ACCURACY (1 Year) 23°C ± 5°C ± (% rdg. + volts)**	**TYPICAL NOISE (peak-peak) 0.1 Hz–10 Hz**
	200.000 mV	5 µV	0.02% + 375 µV	20 µV
	2.00000 V	50 µV	0.02% + 600 µV	50 µV
	20.0000 V	500 µV	0.02% + 5 mV	300 µV
	200.000 V	5 mV	0.02% + 50 mV	2 mV
Temperature Coefficient	± (0.15 × accuracy specification)/°C • For temperatures (0°–18°C & 28°–50°C)			
Maximum Output Power and Source/Sink Limits[2]	30.3 W per channel maximum. • ± 20.2 V @ ± 1.5 A • ± 202 V @ ± 100 mA • Four-quadrant source or sink operation.			
Voltage Regulation	Line: 0.01% of range Load: ± (0.01% of range + 100 µV).			
Noise 10 Hz – 20 MHz	< 20 mV peak-peak (typical), < 3 mV RMS (typical) • 20 V range			
Current Limit/Compliance[3]	Bipolar current limit (compliance) set with single value. Minimum value is 100 pA. Accuracy is the same as current source.			

[1] Add 50 µV 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.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Specifications Category	Specifications
Overshoot	< ± (0.1% + 10 mV) (typical) • Step size = 10% to 90% of range, resistive load, maximum - current limit/compliance.
Guard Offset Voltage	< 4 mV • Current < 10 mA

CURRENT SOURCE SPECIFICATIONS

Specifications Category	Specifications			
	RANGE	**PROGRAMMING RESOLUTION**	**ACCURACY (1 Year) 23°C ± 5°C ± (% rdg. + amps)**	**TYPICAL NOISE (peak-peak) 0.1 Hz–10 Hz**
Current Programming Accuracy	1.00000 nA	20 fA	0.15% + 2 pA	800 fA
	10.0000 nA	200 fA	0.15% + 5 pA	2 pA
	100.000 nA	2 pA	0.06% + 50 pA	5 pA
	1.00000 µA	20 pA	0.03% + 700 pA	25 pA
	10.0000 µA	200 pA	0.03% + 5 nA	60 pA
	100.000 µA	2 nA	0.03% + 60 nA	3 nA
	1.00000 mA	20 nA	0.03% + 300 nA	6 nA
	10.0000 mA	200 nA	0.03% + 6 µA	200 nA
	100.000 mA	2 µA	0.03% + 30 µA	600 nA
	1.00000 A[4]	20 µA	0.05% + 1.8 mA	70 µA
	1.50000 A[4]	50 µA	0.06% + 4 mA	150 µA
Temperature Coefficient	± (0.15 × accuracy specification)/°C • For temperatures (0° – 18°C & 28° – 50°C)			
Maximum Output Power and Source/Sink Limits[4]	30.3 W per channel maximum. • ± 1.515 A @ ± 20 V • ± 101 mA @ ± 200 V • Four-quadrant source or sink operation.			
Current Regulation	Line: 0.01% of range Load: ± (0.01% of range + 100pA).			
Voltage Limit/Compliance[5]	Bipolar voltage limit (compliance) set with single value. Minimum value is 10 mV. Accuracy is the same as current source.			

[3] For sink mode operation (quadrants 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] 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

[5] For sink mode operation (quadrants II and IV), add 10% of compliance range and ±0.02% of limit setting to corresponding voltage source specification. For 200mV range add an additional 120mV of uncertainty.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Specifications Category	Specifications
Overshoot	< ± 0.1% (typical) • step size = 10% to 90% of range, resistive load, maximum - current limit/compliance • See CURRENT SOURCE OUTPUT SETTLING TIME for additional test conditions

ADDITIONAL SOURCE SPECIFICATIONS

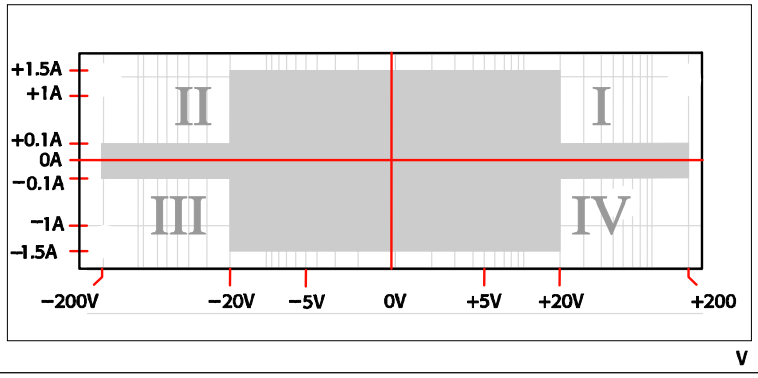
Specifications Category	Specifications	
Transient Response Time	< 70 µs for the output to recover to 0.1% for a 10% to 90% step change in load.	
Voltage Source Output Settling Time	Time required to reach 0.1% of final value after source level command is processed on a fixed range.	
	Range	**Settling Time**
	200 mV	< 50 µs (typical)
	2 V	< 50 µs (typical)
	20 V	< 110 µs (typical)
	200 V	< 700 µs (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 = 2 V unless noted	
	Current Range	**Settling Time**
	1.5 A – 1 A	< 120 µs (typical) (Rload > 6)
	100 mA – 10 mA	< 80 µs (typical)
	1 mA	< 100 µs (typical)
	100 µA	< 150 µs (typical)
	10 µA	< 500 µs (typical)
	1 µA	< 2 ms (typical)
	100 nA	< 20 ms (typical)
	10 nA	< 40 ms (typical)
	1 nA	< 150 ms (typical)
DC Floating Voltage	Output can be floated up to ± 250 VDC	
Remote Sense Operating Range[6]	Maximum voltage between HI and SENSE HI = 3 V Maximum voltage between LO and SENSE LO = 3V	
Voltage Output Headroom	200 V Range • Maximum output voltage = 202.3 V – total voltage drop across source leads. (maximum 1 per source lead) 20 V Range • Maximum output voltage = 23.3 V – total voltage drop across source leads. (maximum 1 per source lead)	

[6] Add 50 µV to source accuracy specifications per volt of HI lead drop.

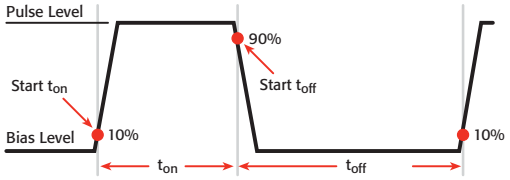
Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Specifications Category	Specifications
Over Temperature Protection	Internally sensed temperature overload puts unit in standby mode.
Voltage Source Range Change Overshoot	300 mV + 0.1% of larger range (typical) • Overshoot into a 200 K load, 20 MHz BW
Current Source Range Change Overshoot	< 5% of larger range + 300 mV/Rload + 60 nA (typical) • See CURRENT SOURCE OUTPUT SETTLING TIME for additional test conditions.

PULSE SPECIFICATIONS

Specifications Category	Specifications
Minimum Programmable Pulse Width[7]	200 µs • Note: Minimum pulse width for settled source at a given I/V output and load can be longer than 200 µs.
Pulse Width Programming Resolution	1 µs
Pulse Width Programming Accuracy	± 25 µs
Pulse Width Jitter	50 µs (typical)
Quadrant Diagram	

7 Times measured from the start of pulse to the start off-time; see figure below.



Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

3. METER SPECIFICATIONS

VOLTAGE MEASUREMENT SPECIFICATIONS

Specifications Category	Specifications			
	RANGE	**DISPLAY RESOLUTION[9]**	**INPUT IMPEDENCE**	**ACCURACY (1 Year) 23°C ± 5°C ± (% rdg. + volts)**
Voltage Measurement Accuracy[8,9]	200.00 mV	1 µV	> 10 GΩ	0.015% + 225 µV
	2.00000 V	10 µV	> 10 GΩ	0.02% + 350 µV
	20.0000 V	100 µV	> 10 GΩ	0.015% + 5 mV
	200.000 V	1 mV	> 10 GΩ	0.015% + 50 mV
Temperature Coefficient	± (0.15 × accuracy specification)/°C • For temperatures (0°–18°C & 28°–50°C)			

CURRENT MEASUREMENT SPECIFICATIONS[9]

Specifications Category	Specifications			
	RANGE	**DISPLAY RESOLUTION[10]**	**VOLTAGE BURDEN[11]**	**ACCURACY (1 Year) 23°C ± 5°C ± (% rdg. + amps)**
Current Measurement Accuracy	100.00 pA[12,13]	1 fA	< 1 mV	0.15% + 120 fA
	1.00000 nA[12,14]	10 fA	< 1 mV	0.15% + 240 fA
	10.0000 nA	100 fA	< 1 mV	0.15% + 3 pA
	100.000 nA	1 pA	< 1 mV	0.06% + 40 pA
	1.00000 µA	10 pA	< 1 mV	0.025% + 400 pA
	10.0000 µA	100 pA	< 1 mV	0.025% +1.5 nA

[8] Add 50µV to source accuracy specifications per volt of HI lead drop

[9] De-rate accuracy specifications for NPLC setting < 1 by increasing error term. Add appropriate % of range term using table below

NPLC Setting	200 mV Range	2 V – 200 V Ranges	100 nA Range	1 µA – 100 mA Ranges	1 A – 1.5 A Ranges
0.1	0.01%	0.01%	0.01%	0.01%	0.01%
0.01	0.08 %	0.07%	0.1 %	0.05%	0.05%
0.001	0.8 %	0.6 %	1 %	0.5 %	1.1 %

[10] Applies when in single channel display mode.

[11] Four-wire remote sense only.

[12] 10-NPLC, 11-Point Median Filter, < 200V range, measurements made within 1 hour after zeroing. 23°C ± 1°C

[13] Under default specification conditions: ±(0.15% + 750 fA).

[14] Under default specification conditions: ±(0.15% + 1 pA).

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Specifications Category	Specifications			
	RANGE	**DISPLAY RESOLUTION[15]**	**VOLTAGE BURDEN[16]**	**ACCURACY (1 Year) 23°C ± 5°C ± (% rdg. + amps)**
	100.000 µA	1 nA	< 1 mV	0.02% + 25 nA
	1.00000 mA	10 nA	< 1 mV	0.02% +200 nA
	10.0000 mA	100 nA	< 1 mV	0.02% + 2.5 µA
	100.000 mA	1 µA	< 1 mV	0.02% +20 µA
	1.00000 A	10 µA	< 1 mV	0.03% +1.5 mA
	1.50000 A	10 µA	< 1 mV	0.05% + 3.5 mA
Current Measure[17] Settling Time	Time required to reach 0.1% of final value:			
	Current Range		**Settling Time**	
	1 mA		< 100 µs (typical)	
Temperature Coefficient	± (0.15 × accuracy specification)/°C • For temperatures (0°–18°C & 28°–50°C)			

ADDITIONAL METER SPECIFICATIONS

Specifications Category	Specifications
Load Impedance	Stable into 10,000 pF (typical)
Common Mode Voltage	250 VDC
Common Mode Isolation	> 1 GΩ < 4500 pF
Overrange	101% of source range 102% of measure range
Maximum Sense Lead Resistance	1 KΩ for rated accuracy
Sense Input Impedance	> 10 GΩ

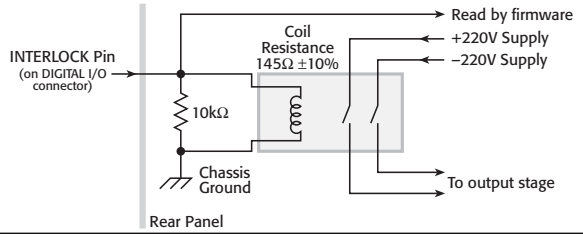
[15] Applies when in single channel display mode.
[16] Four-wire remote sense only.
[17] See series 2600 SourceMeter Reference Manual Section 8 for more information.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

4. GENERAL

Specifications Category	Specifications
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 300bps to 115200bps. 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	See 2601/02 GENERAL specifications for circuit diagram
Connector	25-pin female D
Input/Output Pins	14 open drain I/O bits
Absolute Maximum Input Voltage	5.25 V
Absolute Minimum Input Voltage	– 0.25 V
Maximum Logic Low Input Voltage	0.7V, +850 µA max
Minimum Logic High Input Voltage	2.1V, + 570 µA
Maximum Source Current (flowing out of Digital I/O bit)	+ 960 µA
Maximum Sink Current @ Maximum Logic Low Voltage (0.7V	– 5.0 mA
Absolute Maximum Sink Current (flowing into Digital I/O pin	– 11 mA
5V Power Supply Pin	Limited to 600 mA, solid state fuse protected
Safety Interlock Pin	Active high input. > 3.4 V @ 24 mA (absolute maximum of 6 V) must be externally applied to this pin to insure 200 V operation. This signal is pulled down to chassis ground with a 10 K resistor. 200 V operation will be blocked when the INTERLOCK signal is < 0.4 V (absolute minimum –0.4 V). See figure below:

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

Specifications Category	Specifications
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 300bps to 115200bps. 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
	
Power Supply	100 V to 240 VAC, 50 Hz – 60 Hz (manual setting), 240 VA 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 2004/108/EEC, EN 61326-1
Safety	Conforms to European Union Directive 73/23/EEC, EN 61010-1, and UL 61010-1
Dimensions	89 mm high × 213 mm wide × 460 mm deep (3½ in × 8⅜ in × 17½ in). Bench Configuration (with handle & feet): 104 mm high × 238 mm wide × 460 mm deep (4⅛ in × 9⅜ in × 17½ in)
Weight	2635: 4.75 Kg (10.4 lbs). 2636: 5.50 Kg (12.0 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

KEITHLEY

Series 2600

System SourceMeter® Specifications

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

1. SPEED SPECIFICATIONS[1,2,3]

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	1110 (1000)	880 (880)	840 (840)
0.01 NPLC	Internal	950 (900)	780 (760)	730 (710)
0.1 NPLC	Internal	390 (345)	355 (320)	340 (305)
1.0 NPLC	Internal	57 (48)	56 (47)	56 (47)

Maximum measurement range change rate: >4500/second typical. When changing to or from a range ≥1A, maximum rate is >2000/second typical.

Maximum source range change rate: >400/second, typical.

Maximum source function change rate: >500/second, typical.

External trigger input: The digital I/O interface signals can be configured to operate 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.

---

1 See the Speed Specifications Test Conditions Appendix in the Series 2600 Reference Manual for more information regarding test conditions.
2 Exclude current measurement ranges less than 1mA.
3 2635/2636 with default measurement delays and filters disabled.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

2. SUPPLEMENTAL INFORMATION

Front Panel Interface:

Two-line vacuum fluorescent 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: 3Mb (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

Software Interface: Direct GPIB/VISA, Read/Write with VB, VC/C++, LabVIEW, TestPoint, LabWindows/CVI, etc.

Keithley Instruments, Inc.
28775 Aurora Road
Cleveland, Ohio 44139
1-888-KEITHLEY
www.keithley.com

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:

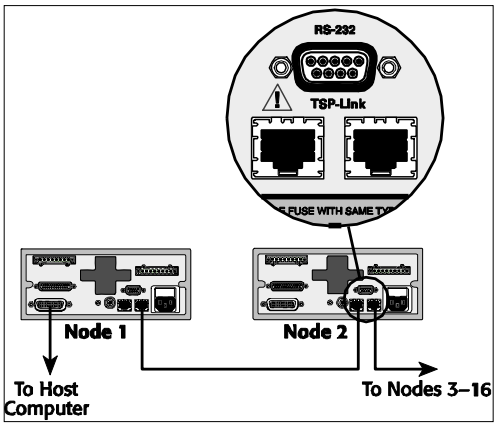
- Measurement
- Measurement status
- Timestamp
- Source setting (at the time the measurement was taken)
- Range information

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. Typical battery life is 1 year.

Factory TSP Scripts:
See 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 SourceMeter instruments 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 16 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: 1µs.
Accuracy: 100ppm.

This page left blank intentionally.

Appendix B
Error and Status Messages

In this appendix:

Topic	Page
Introduction	B-2
Error summary	B-2
Error effects on scripts	B-2
Reading errors	B-2

Introduction

This appendix includes information on Keithley Instruments Series 2600 System SourceMeter® error levels, how to read errors, and a complete listing of error messages.

Error summary

Error messages are listed in Table B-2. Error levels are listed below:

- NO_SEVERITY: informational status message only
- INFORMATIONAL: informational status message only
- RECOVERABLE: error not serious, can be recovered
- SERIOUS: error serious, but unit still operational by correcting error
- FATAL: unit non-operational

Error effects on scripts

Most errors will not abort a running script. The only time a script is aborted is when a Lua run-time error (error number -286) is detected. Run-time errors are caused by actions such as trying to index into a variable that is not a table. Syntax errors (error number -285) in a script/command will not technically abort the script, but it will prevent the script/command from being executed in the first place.

Reading errors

When errors occur, the error messages will be placed in the error queue (see "Queues" in Appendix D). 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 return the message portion of the error:

```
errorcode, message, severity, node = errorqueue.next()

print(message)
```

Table B-1
Error queue commands

Error queue command	Description
`errorqueue.clear()`	Clear error queue of all errors.
`errorqueue.count`	Number of messages in the error/event queue.
`errorqueue.next()`	Request error message.

Table B-2
Error summary

Error number	Error level	Error Message
-430	RECOVERABLE	Query Deadlocked
-420	RECOVERABLE	Query Unterminated
-410	RECOVERABLE	Query Interrupted
-363	RECOVERABLE	Input Buffer Over-run
-350	RECOVERABLE	Queue Overflow
-315	RECOVERABLE	Configuration Memory Lost
-314	RECOVERABLE	Save/ Recall Memory Lost
-292	RECOVERABLE	Referenced name does not exist
-286	RECOVERABLE	TSP Runtime error
-285	RECOVERABLE	Program Syntax
-281	RECOVERABLE	Cannot Create Program
-225	RECOVERABLE	Out of Memory or TSP Memory allocation error
-224	RECOVERABLE	Illegal Parameter Value
-223	RECOVERABLE	Too Much Data
-222	RECOVERABLE	Parameter Data Out of Range
-221	RECOVERABLE	Settings Conflict
-220	RECOVERABLE	Parameter
-203	RECOVERABLE	Command protected
-154	RECOVERABLE	String Too Long
-151	RECOVERABLE	Invalid String Data
-144	RECOVERABLE	Character Data Too Long
-141	RECOVERABLE	Invalid Character Data
-121	RECOVERABLE	Invalid Character In Number
-120	RECOVERABLE	Numeric Data
-109	RECOVERABLE	Missing Parameter
-108	RECOVERABLE	Parameter Not Allowed
-105	RECOVERABLE	Trigger Not Allowed
-104	RECOVERABLE	Data Type
-101	RECOVERABLE	Invalid Character
0	NO_SEVERITY	Queue Is Empty

Table B-2 (cont.)
Error summary

Error number	Error level	Error Message
603	RECOVERABLE	Power On State Lost
702	FATAL	Unresponsive digital FPGA
802	RECOVERABLE	Output Blocked By Interlock
820	RECOVERABLE	Parsing Value
900	FATAL	Internal System
1100	RECOVERABLE	Command Unavailable
1101	RECOVERABLE	Parameter Too Big
1102	RECOVERABLE	Parameter Too Small
1103	RECOVERABLE	Max Greater Than Min
1104	RECOVERABLE	Too many digits for param type
1106	RECOVERABLE	Battery Not Present
1107	RECOVERABLE	Cannot modify factory menu
1108	RECOVERABLE	Menu name does not exist
1109	RECOVERABLE	Menu name already exists
1110	FATAL	Catastrophic analog supply failure
1200	RECOVERABLE	TSPlink initialization failed
1201	RECOVERABLE	TSPlink initialization failed
1202	RECOVERABLE	TSPlink initialization failed
1203	RECOVERABLE	TSPlink initialization failed (possible loop in node chain)
1204	RECOVERABLE	TSPlink initialization failed
1205	RECOVERABLE	TSPlink initialization failed (no remote nodes found)
1206	RECOVERABLE	TSPlink initialization failed
1207	RECOVERABLE	TSPlink initialization failed
1208	RECOVERABLE	TSPlink initialization failed
1209	RECOVERABLE	TSPlink initialization failed
1210	RECOVERABLE	TSPlink initialization failed (node ID conflict)
1211	RECOVERABLE	Node %u is inaccessible
1212	RECOVERABLE	Invalid node ID
1400	RECOVERABLE	Expected at least %d parameters
1401	RECOVERABLE	Parameter %d is invalid
1402	RECOVERABLE	User scripts lost
1403	RECOVERABLE	Factory scripts lost
1404	RECOVERABLE	Invalid byte order
1405	RECOVERABLE	Invalid ASCII precision
1406	RECOVERABLE	Invalid data format
1500	RECOVERABLE	Invalid baud rate setting
1501	RECOVERABLE	Invalid parity setting
1502	RECOVERABLE	Invalid terminator setting
1503	RECOVERABLE	Invalid bits setting
1504	RECOVERABLE	Invalid flow control setting
1600	RECOVERABLE	Maximum GPIB message length exceeded
1800	RECOVERABLE	Invalid Digital Trigger Mode
1801	RECOVERABLE	Invalid Digital I/O Line
2000	RECOVERABLE	Flash download error
2001	RECOVERABLE	Cannot flash with error in queue
4900	RECOVERABLE	Reading buffer index %s is invalid
4901	RECOVERABLE	The maximum index for this buffer is %d
4903	RECOVERABLE	Reading buffer expired

Table B-2 (cont.)
Error summary

Error number	Error level	Error Message
4904	SERIOUS	ICX parameter count mismatch, %s (Line #%d)
4905	SERIOUS	ICX parameter invalid value, %s (Line #%d)
4906	SERIOUS	ICX invalid function id, %s (Line #%d)
5001	FATAL	SMU is unresponsive
5003	SERIOUS	Saved calibration constants corrupted
5004	SERIOUS	Operation conflicts with CALA sense mode
5005	SERIOUS	Value too big for range
5007	SERIOUS	Operation would exceed safe operating area of the instrument
5008	SERIOUS	Operation not permitted while output is on
5009	SERIOUS	Unknown sourcing function
5010	SERIOUS	No such SMU function
5011	SERIOUS	Operation not permitted while cal is locked
5012	SERIOUS	Cal data not saved - save or restore before lock
5013	SERIOUS	Cannot save cal data - unlock before save
5014	SERIOUS	Cannot restore cal data - unlock before restore
5015	SERIOUS	Save to cal set disallowed
5016	SERIOUS	Cannot change cal date - unlock before operation
5017	SERIOUS	Cannot change cal constants - unlock before operation
5018	SERIOUS	Cal version inconsistency
5019	SERIOUS	Cannot unlock - invalid password
5021	SERIOUS	Cannot restore default calset. Using previous calset
5022	SERIOUS	Cannot restore previous calset. Using factory calset
5023	SERIOUS	Cannot restore factory calset. Using nominal calset
5024	SERIOUS	Cannot restore nominal calset. Using firmware defaults
5025	SERIOUS	Cannot set filtercount > 1 when measure.count > 1
5027	SERIOUS	Unlock cal data with factory password
5028	SERIOUS	Cannot perform requested operation while source autorange is enabled
5029	SERIOUS	Cannot save without changing cal date and cal due values
5032	RECOVERABLE	Cannot change this setting unless buffer is cleared
5033	RECOVERABLE	Reading buffer not found within device
5038	RECOVERABLE	Index exceeds maximum reading
5040	RECOVERABLE	Cannot use same reading buffer for multiple overlapped measurements
5041	SERIOUS	Output Enable not asserted
5042	SERIOUS	Invalid while overlapped measure
5043	SERIOUS	Cannot perform requested operation while voltage measure autorange is enabled
5044	SERIOUS	Cannot perform requested operation while current measure autorange is enabled
5045	SERIOUS	Cannot perform requested operation while filter is enabled
5046	SERIOUS	SMU too hot
5047	RECOVERABLE	Minimum timestamp resolution is 1µs
5048	SERIOUS	Contact check not valid with HIGH-Z output off
5049	SERIOUS	Contact check not valid while an active current source
5050	SERIOUS	I limit too low for contact check
5051	FATAL	Model Number/SMU Hardware mismatch
5052	RECOVERABLE	Interlock engaged; system stabilizing
5052	RECOVERABLE	Cannot disable outputenableaction

This page left blank intentionally.

Appendix C
Common Commands

In this appendix:

Topic	Page
Common commands	**C-2**
Command summary	C-2
Script command equivalents	C-2
Command reference	C-3

Common commands

Command summary

Common commands supported by the Keithley Instruments Series 2600 System 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 Series 2600 does not strictly adhere to that standard.

Table C-1
Common commands

Mnemonic	Name	Description[1]
*CLS	Clear status	Clears all event registers and Error Queue.
*ESE <mask>	Event enable command	Program the Standard Event Enable Register.
*ESE?	Event enable query	Read the Standard Event Enable Register.
*ESR?	Event status register query	Read/clear the Standard Event Enable Register.
*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.
*SRE?	Service request enable query	Reads the Service Request Enable Register.
*STB?	Status byte query	Reads the Status Byte Register.[1]
*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)

Table C-2
Script command equivalents

Common command	Script command equivalent
*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, yyyyy

Where:
nnnn is the model number (Model 2601/2602/2611/2612/2635/2636).

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 1-1-5 on page 1-23).

*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 Series 2600 does not actually perform a self-test.

*WAI — wait-to-continue

Wait until commands are completed

Two types of device commands exist:

- Sequential command — A command whose operations are allowed to finish before the next command is executed.
- Overlapped command — A command that allows the execution of subsequent commands while device operations of the overlapped command are still in progress.

The *WAI command is used to suspend the execution of subsequent commands until the device operations of all previous overlapped commands are finished. The *WAI command is not needed for sequential commands.

Appendix D
Status Model

In this appendix:

Topic	Page
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-8**
Programming and reading registers	**D-9**
Programming enable and transition registers	D-9
Reading registers	D-10
Status byte and service request (SRQ)	**D-10**
Status byte register	D-10
Service request enable register	D-12
Serial polling and SRQ	D-12
SPE, SPD (serial polling)	D-12
Status byte and service request commands	D-12
Enable and transition registers	D-13
Controlling node and SRQ enable registers	D-13
Status register sets	**D-15**
System Summary Event Registers	D-15
Standard Event Register	D-16
Operation Event Registers	D-18
Measurement Event Registers	D-24
Register programming example	D-27
Queues	**D-27**
Output queue	D-27
Error queue	D-28
	D-28
TSP-Link system status	**D-28**
Status model configuration example	D-28

Overview

The Keithley Instruments Series 2600 System 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

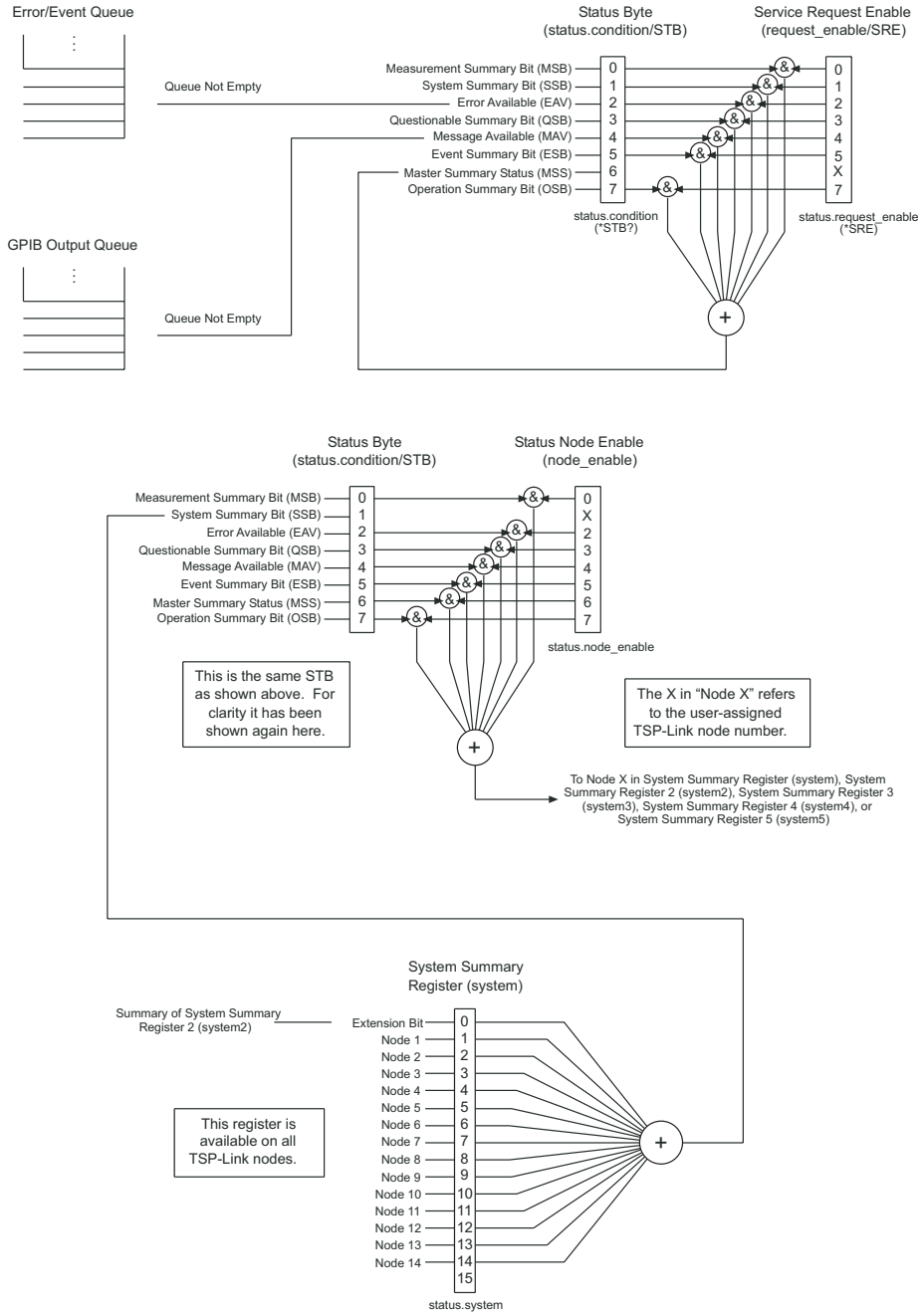


Figure D-2
Status model (system summary and standard event registers)

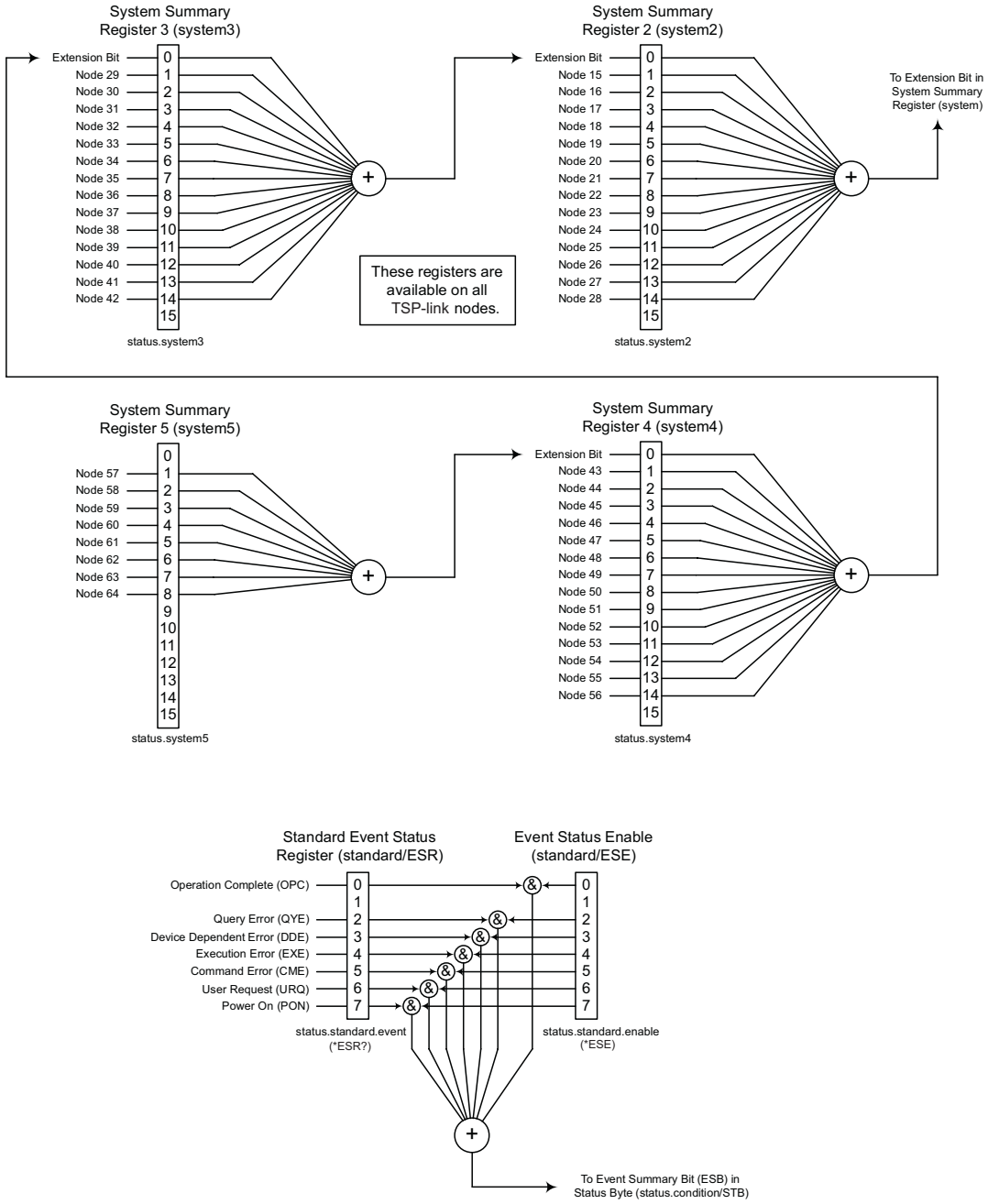


Figure D-3
Status model (operation event registers)

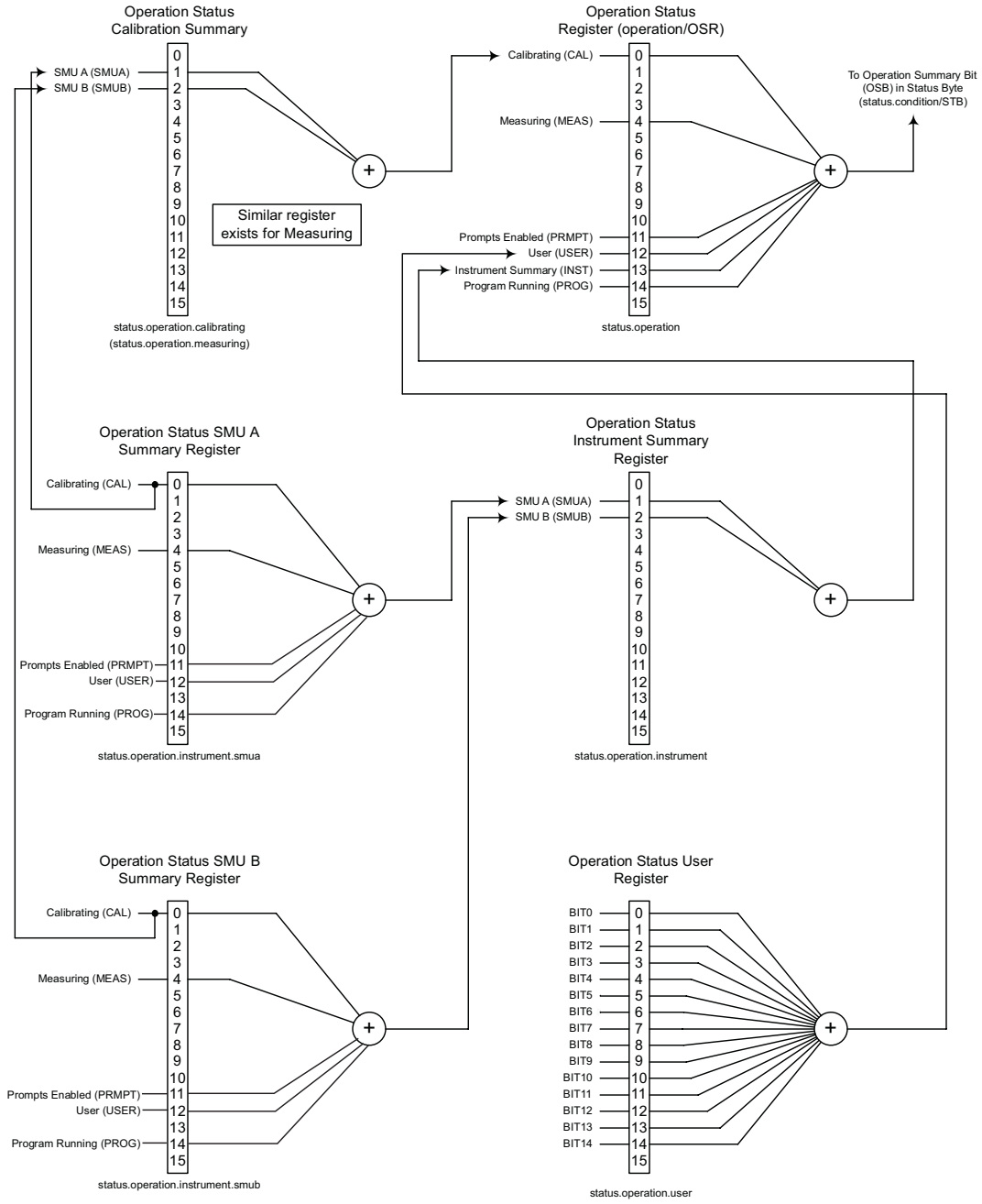


Figure D-4
Status model (questionable event registers)

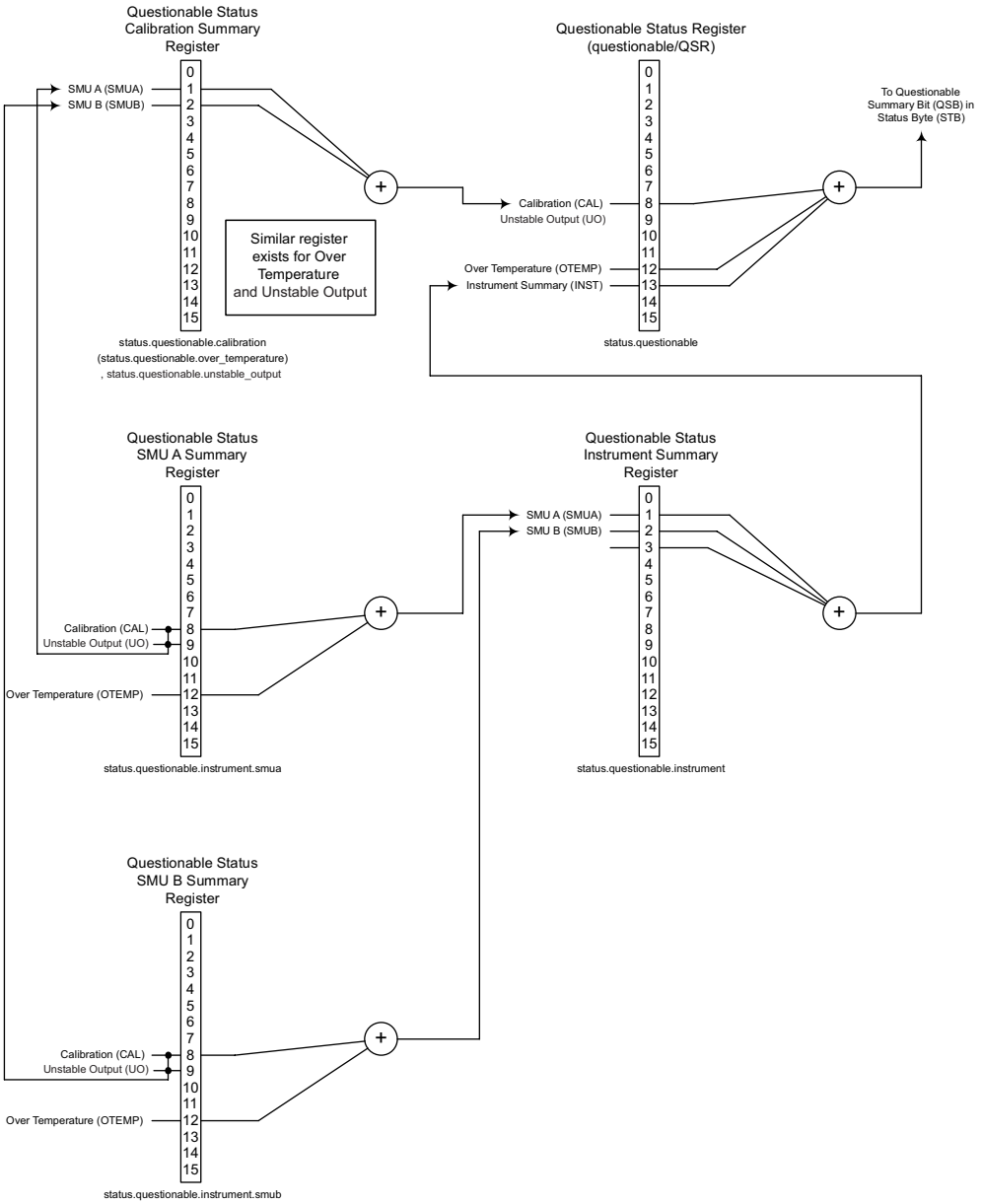
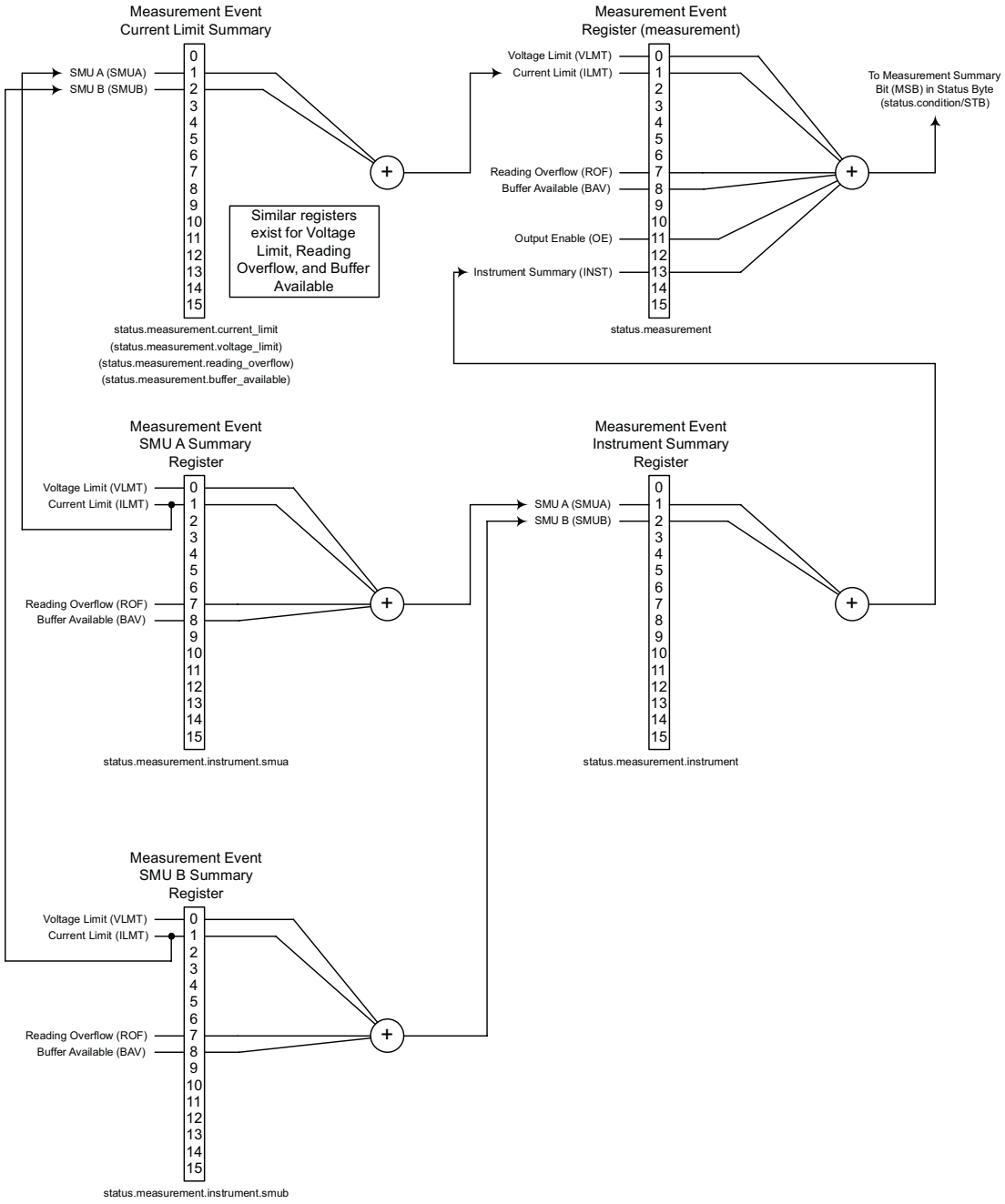


Figure D-5
Status model (measurement event registers)



Status function summary

The following functions control and read the various registers (Table D-1). 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[1]
System summary	status.reset status.node_enable status.request_enable status.node_event status.request_event status.condition
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.*
Operation event	status.operation.* status.operation.instrument.smuX.* status.operation.instrument.* status.operation.calibrating.* status.operation.measuring.* status.operation.user.*
Questionable event	status.questionable.* status.questionable.instrument.smuX.* status.questionable.instrument.* status.questionable.calibration.* status.questionable.over_temperature.* status.questionable.unstable_output.*
Standard event	status.standard.enable
System events	status.system.enable status.system2.enable status.system3.enable status.system4.enable status.system5.enable

1. * = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

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
To Reset Registers:	
	Clears the output queue Reset bits of status registers to 0.
status.reset()	Reset bits of status registers to 0.
To Clear Error Queue:	
	Clear all messages from Error Queue.

The instrument automatically clears the ouput queue when the instrument transitions from the local control state to the remote control state.

Programming and reading registers

Programming enable and transition registers

The only registers that can be programmed by the user are the enable and paragraphs transition registers. All other registers in the status structure are read-only registers. The following explain how to determine the parameter values for the various commands used to program enable registers. The actual commands are summarized in Appendix C and Table D-1.

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" in this appendix, while specific command parameters to program these registers are outlined later in this appendix.

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 Weights	128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)

A. Bits 0 through 7

Bit Position	B15	B14	B13	B12	B11	B10	B9	B8
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal Weights	32768 (2^{15})	16384 (2^{14})	8192 (2^{13})	4096 (2^{12})	2048 (2^{11})	1024 (2^{10})	512 (2^9)	256 (2^8)

B. Bits 8 through 15

When using a numeric parameter, registers are programmed by including the appropriate `<mask>` value, for example:

```
*ese <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.

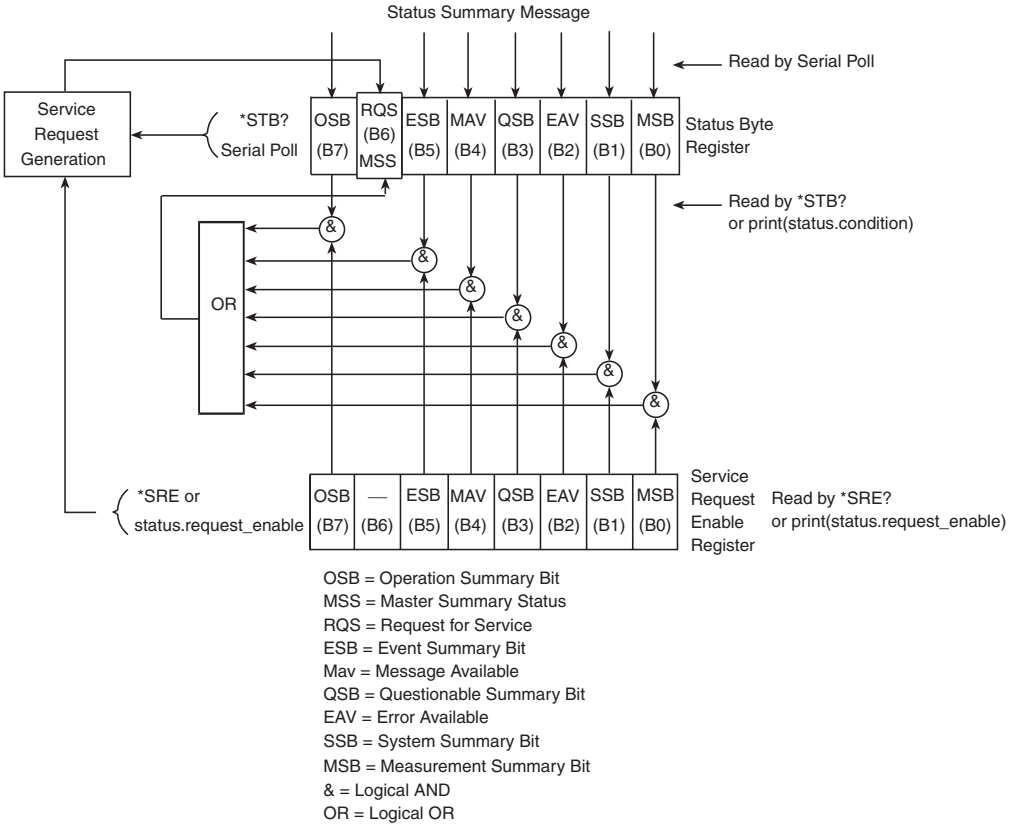
Status byte and service request (SRQ)

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) 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.

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" in Section 11). 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" and "Reading registers" in this appendix.

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
To set system node enable register bits:		
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
To set service request enable register bits:		
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
To read registers:		
print(status.node_enable)	Request system enable register.	
print(status.request_enable)	Request SRQ enable register.	
print(status.condition)	Request status byte register.	

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 and Figure D-2, 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.

In a TSP-Link system, the status model can be configured such that a status event in any node in the system can set the RQS (Request for Service) bit of the Master Node Status Byte. See "'TSP-Link system status'" in this appendix for details on using the status model in a TSP-Link system.

Commands for the system summary registers 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
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see "'Reading registers'" in this appendix. For example, the following command will read the system enable register:

```
print(status.system.enable)
```

The used bits of the system event registers are described as follows:

- **Bit B0, Extension Bit (EXT)** — Set bit indicates that an extension bit from a another system status register is set.
- **Bits B1-B14,[1] NODEn** — Indicates a bit on TSP-Link node n has been set (n = 1 to 64).

1.status.system5 does not use bits B9 through B15.

Standard Event Register

The bits used in the Standard Event Register (shown in Figure D-8) are described as follows:

- **Bit B0, Operation Complete** (OPC) — 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 — The 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-5, and bits are summarized in Table D-6.

Figure D-8
Standard event register

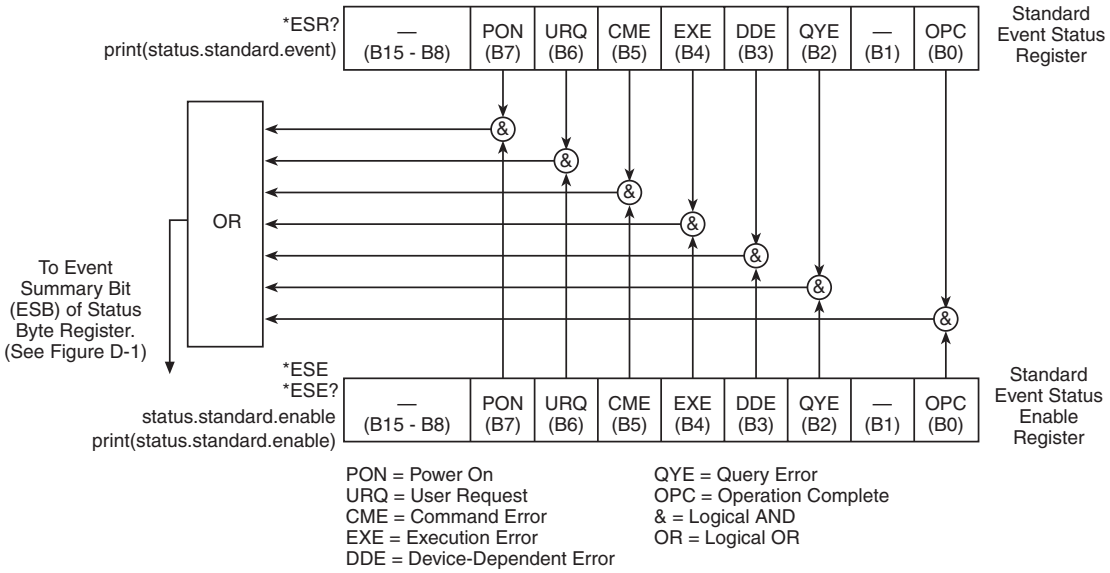


Table D-5
Standard event commands

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 Table D-6.
*ESE? or print(status.standard.enable)	Read Event Status Enable Register.

Table D-6
Status event status registers and bits

Command	Bit
To set register bits:	
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.DDE	B2
status.standard.enable = status.standard.QYE	B3
status.standard.enable = status.standard.DEVICE_DEPENDENT_ERROR	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
To read registers:	
print(status.standard.enable)	
print(status.standard.condition)	
print(status.standard.event)	

Operation Event Registers

As shown in Figure D-3, there are seven register sets associated with Operation Event Status. Commands are summarized in Table D-7. Keep in mind that bits can also be set by using numeric parameter values. For details, see "Programming enable and transition registers" in this appendix.

Table D-7
Operation event commands

Command[1]	Bit
To set register bits:	
status.operation.* = status.operation.CALIBRATING	B0
status.operation.* = status.operation.CAL	B0
status.operation.* = status.operation.MEASURING	B4
status.operation.* = status.operation.MEAS	B4
status.operation.* = status.operation.PROMPTS	B11
status.operation.* = status.operation.PRMPTS	B11
status.operation.* = status.operation.USER	B12
status.operation.* = status.operation.INSTRUMENT_SUMMARY	B13
status.operation.* = status.operation.INST	B13
status.operation.* = status.operation.PROGRAM_RUNNING	B14
status.operation.* = status.operation.PROG	B14

1. * = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.
** = '.ntr', '.ptr', '.enable', and '.condition'; smuX = smua or smub.

Table D-7 (continued)
Operation event commands

Command[1]	Bit
To read registers:	
print(status.operation.*)	
print(status.operation.condition)	
print(status.operation.event)	
To set register bits:	
status.operation.instrument.smuX.* = status.operation.CALIBRATING	B0
status.operation.instrument.smuX.* = status.operation.CAL	B0
status.operation.instrument.smuX.* = status.operation.MEASURING	B4
status.operation.instrument.smuX.* = status.operation.MEAS	B4
status.operation.instrument.smuX.* = status.operation.PROMPTS	B11
status.operation.instrument.smuX.* = status.operation.PRMPTS	B11
status.operation.instrument.smuX.* = status.operation.USER	B12
status.operation.instrument.smuX.* = status.operation.PROGRAM_RUNNING	B14
status.operation.instrument.smuX.* = status.operation.PROG	B14
To read registers:	
print(status.operation.instrument.smuX.*)	
print(status.operation.instrument.smuX.condition)	
print(status.operation.instrument.smuX.event)	
To set register bits:	
status.operation.instrument.* = status.operation.instrument.SMUA	B1
status.operation.instrument.* = status.operation.instrument.SMUB	B2
To read registers:	
print(status.operation.instrument.*)	
print(status.operation.instrument.condition)	
print(status.operation.instrument.event)	
To set register bits:	
status.operation.calibrating.* = status.operation.calibrating.SMUA	B1
status.operation.calibrating.* = status.operation.calibrating.SMUB	B2
To read registers:	
print(status.operation.calibrating.*)	
print(status.operation.calibrating.condition)	
print(status.operation.calibrating.event)	
To set register bits:	
status.operation.measuring.* = status.operation.measuring.SMUA	B1
status.operation.measuring.* = status.operation.measuring.SMUB	B2
To read registers:	
print(status.operation.measuring.*)	
print(status.operation.measuring.condition)	
print(status.operation.measuring.event)	

1. * = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.
** = '.ntr', '.ptr', '.enable', and '.condition'; smuX = smua or smub.

Table D-7 (continued)
Operation event commands

Command[1]	Bit
To set register bits:	
status.operation.user.** = status.operation.user.BIT0	B0
status.operation.user.** = status.operation.user.BIT1	B1
status.operation.user.** = status.operation.user.BIT2	B2
status.operation.user.** = status.operation.user.BIT3	B3
status.operation.user.** = status.operation.user.BIT4	B4
status.operation.user.** = status.operation.user.BIT5	B5
status.operation.user.** = status.operation.user.BIT6	B6
status.operation.user.** = status.operation.user.BIT7	B7
status.operation.user.** = status.operation.user.BIT8	B8
status.operation.user.** = status.operation.user.BIT9	B9
status.operation.user.** = status.operation.user.BIT10	B10
status.operation.user.** = status.operation.user.BIT11	B11
status.operation.user.** = status.operation.user.BIT12	B12
status.operation.user.** = status.operation.user.BIT13	B13
status.operation.user.** = status.operation.user.BIT14	B14
To read registers:	
print(status.operation.user.*)	
print(status.operation.user.condition)	
print(status.operation.user.enable)	

1. * = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.
** = '.ntr', '.ptr', '.enable', and '.condition'; smuX = smua or smub.

For example, either of the following commands will set the CAL enable bit (B0):

```
status.operation.enable = status.operation.CAL
```

```
status.operation.enable = 1
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see "'Reading registers'" in this appendix. For example, the following command will read the operation enable register:

```
print(status.operation.enable)
```

Operation Status Register

This register set feeds to bit B7 (OSB) of the Status Byte. The bits used in the Operation Status Register set are described as follows:

- **Bit B0, Calibrating (CAL)** — Set bit indicates that one or more channels are calibrating.
- **Bit B4, Measuring (MEAS)** — Bit will be set when taking an overlapped measurement, but it will not set when taking a normal synchronous measurement.

- **Bit B11, Prompts Enabled (PRMPT)** — Set bit indicates that command prompts are enabled.
- **Bit B12, User (USER)** — Set bit indicates that an enabled bit in the operation status user register is set.
- **Bit B13, Instrument Summary (INST)** — Set bit indicates that an enabled bit in the operation status instrument summary register is set.
- **Bit B14, Program Running (PROG)** — Set bit indicates that a program is running.

Operation Status Calibration Summary

This calibration summary register set feeds to CAL bit B0 of the Operation Status Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that a SMU (A or B) is calibrating.

Operation Status Measurement Summary

This measurement summary register set feeds to MEAS bit B4 of the Operation Status Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that a SMU (A or B) is performing an overlapped measurement.

Operation Status SMU A Summary Register

This SMU summary register set feeds to bit B1 (SMUA) of the Operation Status Instrument Summary Register. Used summary bits for this register include B0 (CAL), B4 (MEAS), B11 (PRMPT), B12 (USER) and B14 (PROG). A set bit indicates that the corresponding operation event for SMU A has occurred.

Operation Status SMU B Summary Register

This SMU summary register set feeds to bit B1 (SMUB) of the Operation Status Instrument Summary Register. Used summary bits for this register include B0 (CAL), B4 (MEAS), B11 (PRMPT), B12 (USER) and B14 (PROG). A set bit indicates that the corresponding operation event for SMU B has occurred.

Operation Status Instrument Summary Register

This summary register is fed from the SMU summary registers and then feeds to bit B13 (INST) of the Operation Status Summary Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that an operation event has occurred for SMU A and/or SMU B.

Operation Status User Register

This register set is controlled exclusively by the user. This register set feeds to bit B12 (USER) of the Operation Status Summary Register. Bits of the user event register are set by setting the corresponding bits of the user enable register and the user condition register. For example, the following code will set B1 (Bit 1) of the Operation Status User Register and B12 (USER) of the Operation Status Register:

```
status.operation.user.enable = 2

status.operation.user.condition = 2
```

Questionable Event Registers

As shown in Figure D-4, there are six register sets associated with Questionable Event Status. Commands are summarized in Table D-8. Keep in mind that bits can also be set by using numeric parameter values. For details, see ""Programming enable and transition registers"" in this appendix.

For example, either of the following commands will set the CAL enable bit (B8):

```
status.questionable.enable = status.questionable.CAL
```

```
status.questionable.enable = 256
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see ""Reading registers"" in this appendix. For example, the following command will read the questionable enable register:

```
print(status.questionable.enable)
```

Questionable Status Register

This register set feeds to bit B3 (QSB) of the Status Byte. The bits used in the Questionable Status Register set are described as follows:

- **Bit B8, Calibration (CAL)** — Set bit indicates that calibration is questionable.
- **Bit B9, Unstable Output (UO**) -- Set bit indicates that an unstable output condition was detected.
- **Bit B12, Over Temperature (OTEMP)** — Set bit indicates that an over temperature condition was detected.
- **Bit B13, Instrument Summary (INST)** — Set bit indicates that a bit in the questionable instrument summary register is set.

Questionable Status Calibration Summary

This calibration summary register set feeds to CAL bit B8 of the Questionable Status Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that calibration for a SMU (A or B) is questionable.

Questionable Status Over Temperature Summary

This over temperature summary register set feeds to OTEMP bit B12 of the Questionable Status Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that an over temperature condition was detected for a SMU (A or B).

Questionable Status SMU A Summary Register

This SMU summary register set feeds to bit B1 (SMUA) of the Questionable Status Instrument Summary Register. Used summary bits for this register include B8 (CAL) and B12 (OTEMP). A set bit indicates that the corresponding questionable event for SMU A has occurred.

Questionable Status SMU B Summary Register

This SMU summary register set feeds to bit B2 (SMUB) of the Questionable Status Instrument Summary Register. Used summary bits for this register include B8 (CAL) and B12 (OTEMP). A set bit indicates that the corresponding questionable event for SMU B has occurred.

Questionable Status Instrument Summary Register

This summary register is fed from the SMU summary registers and then feeds to bit B13 (INST) of the Questionable Status Summary Register. Used summary bits for this register include B1

(SMUA) and B2 (SMUB). A set bit indicates that a questionable event has occurred for SMU A and/or SMU B.

Table D-8
Operation event commands

Command[1]	Bit
To set register bits:	
status.questionable.* = status.questionable.CALIBRATION	B8
status.questionable.* = status.questionable.CAL	B8
status.questionable.* = status.questionable.OVER_TEMPERATURE	B12
status.questionable.* = status.questionable.OTEMP	B12
status.questionable.* = status.questionable.INSTRUMENT_SUMMARY	B13
status.questionable.* = status.questionable.INST	B13
To read registers:	
print(status.questionable.*)	
print(status.questionable.condition)	
print(status.questionable.event)	
To set register bits:	
status.questionable.instrument.smuX.* = status.questionable.CALIBRATION	B8
status.questionable.instrument.smuX.* = status.questionable.CAL	B8
status.questionable.instrument.smuX.* = status.questionable.OVER_TEMPERATURE	B12
status.questionable.instrument.smuX.* = status.questionable.OTEMP	B12
To read registers:	
print(status.questionable.instrument.smuX.*)	
print(status.questionable.instrument.smuX.condition)	
print(status.questionable.instrument.smuX.event)	
To set register bits:	
status.questionable.instrument.* = status.questionable.instrument.SMUA	B1
status.questionable.instrument.* = status.questionable.instrument.SMUB	B2
To read registers:	
print(status.questionable.instrument.*)	
print(status.questionable.instrument.condition)	
print(status.questionable.instrument.event)	
To set register bits:	
status.questionable.calibration.* = status.questionable.calibration.SMUA	B1
status.questionable.calibration.* = status.questionable.calibration.SMUB	B2
To read registers:	
print(status.questionable.calibration.*)	
print(status.questionable.calibration.condition)	
print(status.questionable.calibration.event)	
To set register bits:	
status.questionable.over_temperature.* = status.questionable.over_temperature.SMUA	B1
status.questionable.over_temperature.* = status.questionable.over_temperature.SMUB	B2
To read registers:	
print(status.questionable.over_temperature.*)	
print(status.questionable.over_temperature.condition)	
print(status.questionable.over_temperature.event)	
To set register bits:	
status.questionable.unstable_output.* = status.questionable.unstable_output.SMUA	B1
status.questionable.unstable_output.* = status.questionable.unstable_output.SMUB	B2
To read registers:	
print(status.questionable.unstable_output.*)	
print(status.questionable.unstable_output.condition)	

Table D-8 (continued)
Operation event commands

Command[1]	Bit
print(status.questionable.unstable_output.event)	

1. * = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

Measurement Event Registers

As shown in Figure D-5, there are seven register sets associated with Measurement Event Status. Commands are summarized in Table D-9. Keep in mind that bits can also be set by using numeric parameter values. For details, see ""Programming enable and transition registers"" in this appendix.

For example, either of the following commands will set the VOLTAGE_LIMIT enable bit:

```
status.measurement.enable = status.measurement.VOLTAGE_LIMIT
```

```
status.measurement.enable = 1
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see ""Reading registers"" in this appendix. For example, the following command will read the measurement enable register:

```
print(status.measurement.enable)
```

The bits used in the Measurement Event Registers are described as follows:

- **Bit B0, Voltage Limit (VLMT)** — Set bit indicates that the voltage limit was exceeded. This bit will be updated only when (1) a measurement is taken or (2) the `smuX.source.compliance` command is invoked.
- **Bit B1, Current Limit (ILMT)** — Set bit indicates that the current limit was exceeded. This bit will be updated only when (1) a measurement is taken or (2) the `smuX.source.compliance` command is invoked.
- **Bit B7, Reading Overflow (ROF)** — Set bit indicates that an overflow reading has been detected.
- **Bit B8, Buffer Available (BAV)** — Set bit indicates that there is at least one reading stored in either or both of the non-volatile reading buffers.
- **Bit B11, Output Enable (OE)** — Set bit indicates that output enable has been asserted.
- **Bit B13, Instrument Summary (INST)** — Set bit indicates that a bit in the measurement instrument summary register is set.

Measurement Event Current Limit Summary

This summary register set feeds to ILMT bit B1 of the Measurement Event Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that a SMU (A or B) is in current limit (compliance).

Measurement Event Voltage Limit Summary

This summary register set feeds to VLMT bit B0 of the Measurement Event Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that a SMU (A or B) is in voltage limit (compliance).

Measurement Event Reading Overflow Summary

This summary register set feeds to ROF bit B7 of the Measurement Event Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that a reading overflow has occurred for a SMU (A or B).

Measurement Event Buffer Available Summary

This summary register set feeds to BAV bit B8 of the Measurement Event Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that a buffer is available for a SMU (A or B).

Measurement Event SMU A Summary Register

This SMU summary register set feeds to bit B1 (SMUA) of the Measurement Status Instrument Summary Register. Used summary bits for this register include B0 (VLMT), B1 (ILMT), B7 (ROF) and B8 (BAV). A set bit indicates that the corresponding measurement event for SMU A has occurred.

Measurement Event SMU B Summary Register

This SMU summary register set feeds to bit B2 (SMUB) of the Measurement Status Instrument Summary Register. Used summary bits for this register include B0 (VLMT), B1 (ILMT), B7 (ROF) and B8 (BAV). A set bit indicates that the corresponding measurement event for SMU B has occurred.

Measurement Event Instrument Summary Register

This summary register is fed from the SMU summary registers and then feeds to bit B13 (INST) of the Measurement Event Summary Register. Used summary bits for this register include B1 (SMUA) and B2 (SMUB). A set bit indicates that an measurement event has occurred for SMU A and/or SMU B.

Table D-9
Operation event commands

Command[1]	Bit
To set register bits:	
status.measurement.* = status.measurement.VOLTAGE_LIMIT	B0
status.measurement.* = status.measurement.VLMT	B0
status.measurement.* = status.measurement.CURRENT_LIMIT	B1
status.measurement.* = status.measurement.ILMT	B1
status.measurement.* = status.measurement.READING_OVERFLOW	B7
status.measurement.* = status.measurement.ROF	B7
status.measurement.* = status.measurement.BUFFER_AVAILABLE	B8
status.measurement.* = status.measurement.BAV	B8
status.measurement.* = status.measurement.OUTPUT_ENABLE	B11
status.measurement.* = status.measurement.OE	B11
status.measurement.* = status.measurement.INSTRUMENT_SUMMARY	B13
status.measurement.* = status.measurement.INST	B13
To read registers:	
print(status.measurement.*)	
print(status.measurement.condition)	
print(status.measurement.event)	

Table D-9 (continued)
Operation event commands

Command[1]	Bit
To set register bits:	
status.measurement.instrument.smuX.* = status.measurement.VOLTAGE_LIMIT	B0
status.measurement.instrument.smuX.* = status.measurement.VLMT	B0
status.measurement.instrument.smuX.* = status.measurement.CURRENT_LIMIT	B1
status.measurement.instrument.smuX.* = status.measurement.ILMT	B1
status.measurement.instrument.smuX.* = status.measurement.READING_OVERFLOW	B7
status.measurement.instrument.smuX.* = status.measurement.ROF	B7
status.measurement.instrument.smuX.* = status.measurement.BUFFER_AVAILABLE	B8
status.measurement.instrument.smuX.* = status.measurement.BAV	B8

1. * = '.ntr', '.ptr' and '.enable'; smuX = smua or smub.

To read registers:	
print(status.measurement.instrument.smuX.*)	
print(status.measurement.instrument.smuX.condition)	
print(status.measurement.instrument.smuX.event)	
To set register bits:	
status.measurement.instrument.* = status.measurement.instrument.SMUA	B1
status.measurement.instrument.* = status.measurement.instrument.SMUB	B2
To read registers:	
print(status.measurement.instrument.*)	
print(status.measurement.instrument.condition)	
print(status.measurement.instrument.event)	
To set register bits:	
status.measurement.voltage_limit.* = status.measurement.voltage_limit.SMUA	B1
status.measurement.voltage_limit.* = status.measurement.voltage_limit.SMUB	B2
To read registers:	
print(status.measurement.voltage_limit.*)	
print(status.measurement.voltage_limit.condition)	
print(status.measurement.voltage_limit.event)	
To set register bits:	
status.measurement.current_limit.* = status.measurement.current_limit.SMUA	B1
status.measurement.current_limit.* = status.measurement.current_limit.SMUB	B2
To read registers:	
print(status.measurement.current_limit.*)	
print(status.measurement.current_limit.condition)	
print(status.measurement.current_limit.event)	
To set register bits:	
status.measurement.buffer_available.* = status.measurement.buffer_available.SMUA	B1
status.measurement.buffer_available.* = status.measurement.buffer_available.SMUB	B2
To read registers:	
print(status.measurement.buffer_available.*)	
print(status.measurement.buffer_available.condition)	

Table D-9 (continued)
Operation event commands

Command[1]	Bit
print(status.measurement buffer_available.event)	
To set register bits:	
status.measurement.reading_overflow.* = status.measurement.reading_overflow.SMUA	B1
status.measurement.reading_overflow.* = status.measurement.reading_overflow.SMUB	B2
To read registers:	
print(status.measurement.reading_overflow.*)	
print(status.measurement.reading_overflow.condition)	
print(status.measurement.reading_overflow.event)	

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 in all 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.system_enable = status.MSB               - Set system summary;
                                                  enable MSB.

status.request_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 in Appendix B).

The SourceMeter status model (Figure D-1) 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-10. 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.

Table D-10
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.

TSP-Link system status

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. In a TSP-Link system, one node (instrument) is the Master and the other nodes are the Slaves. The Master can control the other nodes (Slaves) in the system. See Section 9 for details on the TSP-Link.

The system summary registers (shown in Figure D-1 and Figure D-2) are shared by all nodes in the TSP-Link system. A status event that occurs at a Slave node can generate an SRQ (service request) in the Master node. After detecting the service request, your program can then branch to an appropriate subroutine that will service the request. See "Status byte and service request (SRQ)" in this appendix for details.

Status model configuration example

Figure D-9 shows an example status model configuration for a TSP-Link system. In this example, a current limit (compliance) event in SMU A or SMU B of Node 15 will set the RQS bit of the Status Byte of the Master Node. The commands to configure the status model for this example are provided in ""Status configuration (enable) commands"" in this appendix.

When a current limit (compliance) condition occurs in SMU A or SMU B of Node 15, the following sequence of events will occur:

- **Node 15** – Bit B1 or B2 of the Measurement Event Current Limit Summary register sets when the current limit (compliance) event occurs.
- **Node 15** – Bit B1 (ILMT) of the Measurement Event Register sets.
- **Node 15** – Bit B0 (MSB) of the Status Byte sets.

- **System Summary Registers** – Bit B1 (Node 15) of the System2 Summary Register sets.

NOTE The System Summary Registers are shared by all nodes in the TSP-Link system. When a bit in a system register of Node 15 sets, the same bit in the Master Node system register also sets.

- **System Summary Registers** – Bit B0 (Extension) of the System Summary Register sets.
- **Master Node** – Bit B0 (MSB) of the Status Byte sets.
- **Master Node** – With service request enabled, bit B6 (RQS) of the Status Byte sets. When your program performs the next serial poll of the Master Node, it will detect the current limit event and can branch to a routine to service the request.

Status configuration (enable) commands

The following commands (sent from the Master Node) enable the appropriate register bits for the above example:

Node 15 status registers – The following commands enable the current limit events for SMU A and SMU B of Node 15:

```
[node15].status.measurement.current_limit.enable = 6
```

```
[node15].status.measurement.enable = 2
```

```
[node15].status.node_enable = 1
```

The affected status registers for the above commands are indicated by labels A, B and C in Figure D-9 on page D-30.

System registers – The following commands enable the required system summary bits for Node 15:

```
status.system2.enable = 2
```

```
status.system.enable = 1
```

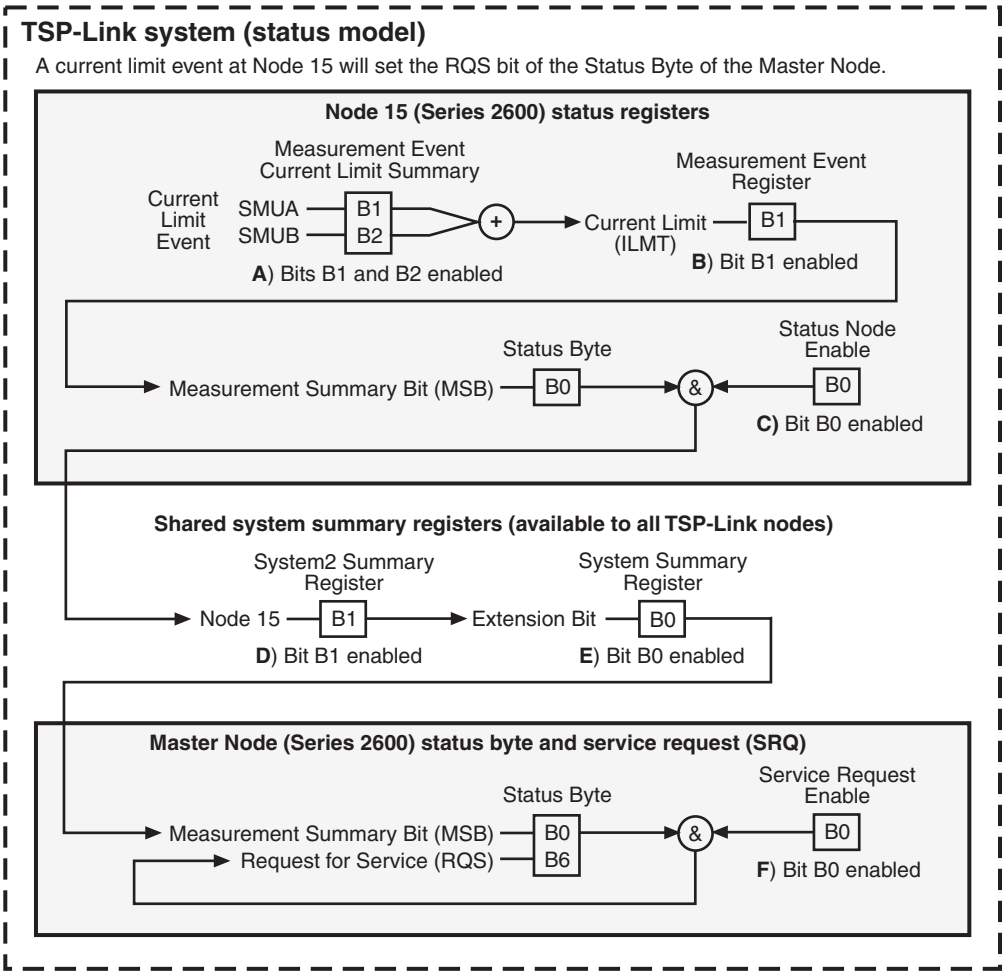
The affected system registers for the above commands are indicated by labels D and E in Figure D-9 on page D-30.

Master Node service request – The following command enables the service request for the measurement event:

```
status.request_enable = 1
```

The affected status register for the above command is indicated by label E in Figure D-9.

Figure D-9
TSP-Link status model configuration example



Appendix E

Speed Specification Test Conditions

In this appendix:

Topic	Page
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-3
Sweep Operation Rates	**E-3**
Digital I/O handshaking:	E-4
Measure to Memory	E-4
Measure to GPIB	E-5
Source Measure to Memory	E-5
Source Measure to GPIB	E-5
Source Measure Pass/Fail to Memory	E-5
Source Measure Pass/Fail to GPIB	E-5
Single Measurement Rates	**E-5**
Measure to GPIB	E-6
Source Measure to GPIB	E-6
Source Measure Pass/Fail to GPIB	E-6
Function/ Range Change Rates	**E-6**
Source Range Change Rate	E-7
Measure Range Change Rate	E-7
Function Change Rate	E-7
Command Processing	**E-7**

Introduction

The purpose of this Keithley Instruments appendix is to provide a general procedure for obtaining speed results similar to those listed in the Series 2600 System SourceMeter® Specifications in Appendix A. Tests were performed using Visual Basic and VISA calls.

Test system used

PC Hardware:	Pentium® 4 2.4GHz, 512MB RAM, National Instruments PCI-GPIB
Series 2600 Unit:	Model 2602 (test runs on smuA only)
Software:	Microsoft® Windows® 2000, Microsoft® Visual Basic® 6.0, VISA version 3.1
Drivers:	(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 the 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 Series 2600 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. The 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, Return-
Count)
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 Series 2600 into a known test state maximizing performance of the instrument. The script 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 Series 2600 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.

NOTE All range change rates are specified as typical. This means that the specification applies to single range steps between adjacent ranges (e.g. 10mA to 100mA). The current (amps) range change specifications exclude ranges < 100mA due to large settling times associated with these low current ranges.

Command Processing

The procedure for a **Command Processing** test is listed below:

1. Using an oscilloscope, connect a probe to the 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 Series 2600 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.”

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                                    --Number of samples to take
    TransmitBuffer = "TestScript()" & CRLF
    VisaStatus = viWrite(VisaSession,
              TransmitBuffer,
              Len(TransmitBuffer), ReturnCount)
    For i = 1 To NumPoints
        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 the Measure to Memory test script.

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 test script 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 **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 =                                    --NumPoints is the number of
"printnumber(smua.measure.v())" & CRLF                samples taken
tlen = Len(TransmitBuffer)

tstart = Timer
   For i = 1 To NumPoints
      VisaStatus = viWrite(VisaSession,
      TransmitBuffer, tlen, ReturnCount)
      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 GPIB

This test is similar to the Measure to GPIB test script. 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 test alternates between two measurement ranges.

Function Change Rate

This test method is the same as the Source Range Change Rate test, but this test 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.

This page left blank intentionally.

Appendix F

Display Character Codes

In this appendix:

Topic	Page
Introduction	F-2
Display character codes (decimal 0-143)	F-2
Display character codes (decimal 144-255)	F-4
Display character dot patterns	F-5

Introduction

This appendix provides a list of display character codes:.

Table F-1
Display character codes (decimal 0-143)

DECIMAL	DISPLAY	DECIMAL	DISPLAY	DECIMAL	DISPLAY
000	reserved	048	0	096	` (open quote)
001	reserved	049	1	097	a
002	reserved	050	2	098	b
003	reserved	051	3	099	c
004	reserved	052	4	100	d
005	reserved	053	5	101	e
006	reserved	054	6	102	f
007	reserved	055	7	103	g
008	reserved	056	8	104	h
009	reserved	057	9	105	i
010	reserved	058	:	106	j
011	reserved	059	;	107	k
012	reserved	060	<	108	l
013	reserved	061	=	109	m
014	reserved	062	>	110	n
015	reserved	063	?	111	o
016	m	064	@	112	p
017	?	065	A	113	q
018	W	066	B	114	r
019	?	067	C	115	s
020	leftflagbar[1]	068	D	116	t
021	rightflagbar	069	E	117	u
022	fullflagbar	070	F	118	v
023	leftbar	071	G	119	w
024	rightbar	072	H	120	x
025	fullflagbar	073	I	121	y
026	?	074	J	122	z
027	?	075	K	123	{

[1] The dot patterns for leftflagbar, rightflagbar, fullflagbar, leftbar, rightbar, and fullflagbar characters found after Table F-2.

Table F-1 (continued)
Display character codes (decimal 0-143)

DECIMAL	DISPLAY	DECIMAL	DISPLAY	DECIMAL	DISPLAY
028	?	076	L	124 |	
029	?	077	M	125 }	
030	selftest1[2]	078	N	126 ~	
031	selftest2	079	O	127 ¦	
032	space	080	P	128	space
033	!	080	Q	129	dot1[3]
034	"	082	R	130	dot2
035	#	083	S	131	dot12
036	$	084	T	132	dot3
037	%	085	U	133	dot13
038	&	086	V	134	dot23
039	' (apostrophe)	087	W	135	dot123
040	(	088	X	136	dot4
041	)	089	Y	137	dot14
042	*	090	Z	138	dot24
043	+	091	[	139	dot124
044	, (comma)	092	\	140	dot34
045	-	093	]	141	dot134
046	.	094	^	141	dot234
047	/	095	_	143	dot1234

[2] The dot patterns for the selftest1 and selftest2 can be found after Table F-2.

[3] The dot patterns for the dot characters can be found after Table F-2.

Table F-2
Display character codes (decimal 144-255)

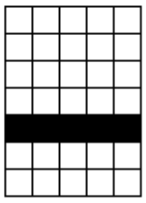
DECIMAL	DISPLAY	DECIMAL	DISPLAY	DECIMAL	DISPLAY
144	selftest3[1]	192	?	240	ÿ
145	selftest4	193		241	lightning
146	selftest5	194	?	242	KI
147	selftest6	195	?	243	box3
148	selftest7	196	?	244	box4
149	selftest8	197	?	245	box5
150	selftest9	198	?	246	box6
151	selftest10	199	?	247	box7
152	selftest11	200	?	248	box8
153	selftest12	201	?	249	box9
154	selftest13	202	?	250	tombstone
155	selftest14	203	?	251	?
156	box1[2]	204	?	252	?
157	box2	205	?	253	?
158	.5[3]	206	Ç	254	?
159	?	207	ç	255	?
160	0	208	æ		
161	1	209	Æ		
162	2	210	â		
163	3	211	ä		
164	4	212	å		
165	5	213	á		
166	6	214	à		
167	7	215	?		
168	8	216	Ä		
169	9	217	Å		
170	a	218	ê		
171	b	219	ë		
172	g	220	é		
173	d	221	è		
174	e	222	É		
175	h	223	î		
176	q	224	ï		
177	l	225	í		
178	p	226	ì		
179	r	227	ô		
180	s	228	ö		
181	t	229	ó		
182	f	230	ò		
183	w	231	?		
184	G	232	Ö		
185	D	233	û		
186	S	234	ü		
187	F	235	ú		
188	?	236	ù		
189	?	237	Ü		
190	?	238	ñ		
191	?	239	Ñ		

1 The dot patterns for the selftest characters can be found after this table.

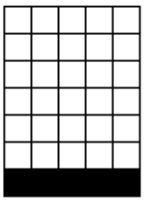
1 The dot patterns for the box characters can be found after this table.

1 The dot patterns for this character can be found after this table.

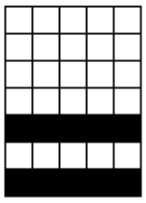
Display character dot patterns



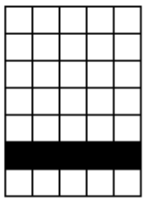
selftest7



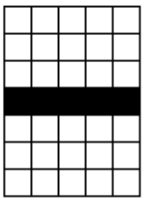
selftest9



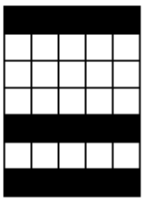
dot34



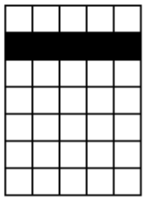
selftest8



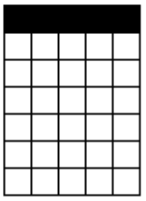
selftest6



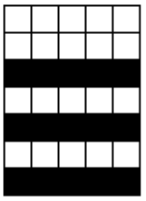
dot134



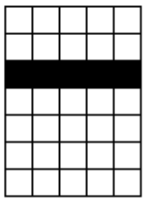
selftest4



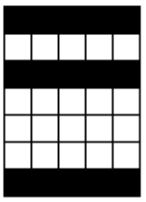
selftest3



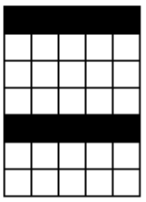
dot234



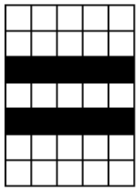
selftest5



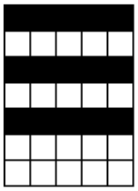
dot124



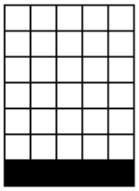
dot13



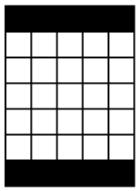
dot23



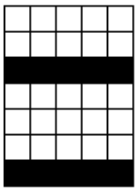
dot123



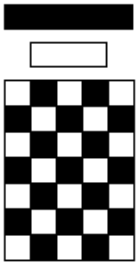
dot4



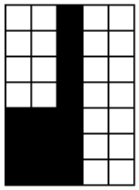
dot14



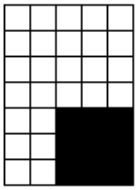
dot24



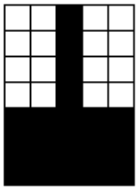
selftest2



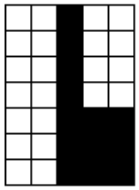
left flag bar



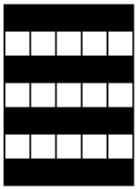
right bar



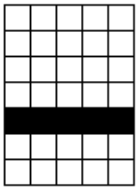
full flag bar



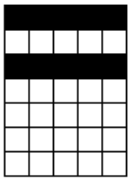
right flag bar



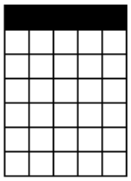
dot1234



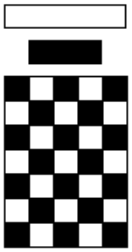
dot3



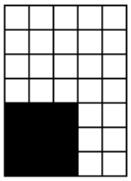
dot12



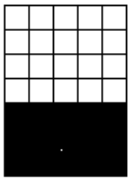
dot1



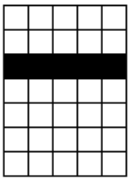
selftest1



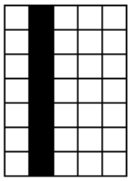
left bar



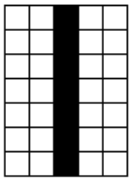
fullbar



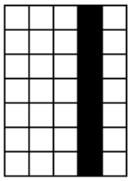
dot2



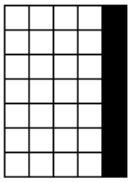
selftest11



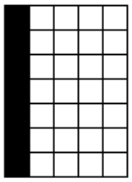
selftest12



selftest13

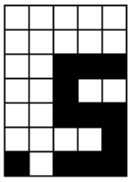


selftest14

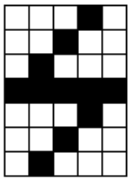


selftest10

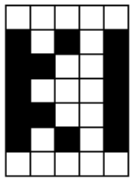
box1



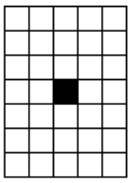
one-half



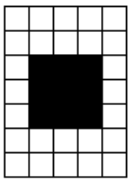
lightning bolt



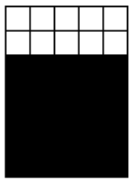
Keithley Instruments



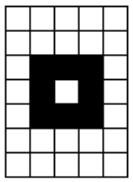
box3



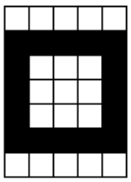
box4



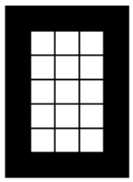
box2



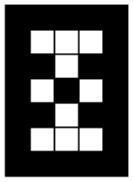
box5



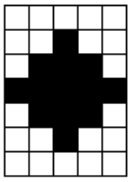
box6



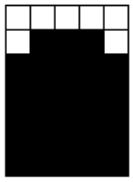
box7



box8



box9



tombstone

Index

A

Accessories 1-4
Attributes 12-3, 12-8
Auto ohms measurements 4-12
Auto range 6-3
Auto zero 4-6
  Front panel 4-7
  Front panel operation 4-7
  NPLC caching 4-6
  Remote command 4-7
  Remote operation 4-7

B

Baud rate 11-9
Beeper 1-15
Branching 2-50
Buffer
  Commands 7-4
  Configuration 7-2
  Dynamically allocated 7-8
  Front panel 7-2
  Location number 7-3
  Overview 7-2
  Programming examples 7-9
  Reading attributes 7-6
  Readings 7-4
  Remote 7-4
  Status 7-8
  Storage attributes 7-5
  Timestamp 7-3

C

Calibration
  Commands 16-7
  Considerations 16-2
  Cycle 16-3
  Errors 16-5
  Procedure 16-8
  Recommended equipment 16-3
  Steps 16-5
Capabilities 1-2
  Source-measure 4-2
Chassis ground 1-10, 1-12, 3-3
Chunk 2-6
Circuit configurations 4-5, 8-16
  Basic 4-5
  Contact check 8-18
  Measure only (V or I) 8-17
  Source I 8-16
  Source V 8-17
Command programming 12-2
  Attributes 12-3, 12-8
  Conventions 12-2
  Functions 12-3, 12-8
  Logical instruments 12-5
  Reading buffers 12-5
  Syntax rules 12-4
  Time and date values 12-7
  TSP-Link nodes 12-4
Commands
  Requesting settings 1-24
Common commands C-2, E-7
  *IDN? C-3
  *OPC C-3
  *OPC? C-3
  *RST C-4
  *TRG C-4
  *TST? C-4
  *WAI C-4
  Summary C-2
Compliance
  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 2-49
Connection
  GPIB 11-2
Connectors
  Digital I/O 1-10, 1-12
  IEEE-488 ..... 1-10, 1-12
  Output 1-10, 1-12
  Power module 1-10, 1-12
  RS-232 ..... 1-10, 1-12
  TSP-Link 1-10, 1-12
Contact check
  Circuit configuration 8-18
  Commands 4-18
  Connections 3-8
  Measurements 4-17
  Overview 4-17
  Programming example 4-19
Contact information 1-3
Continuous power operating boundaries 8-8
Cooling vents 1-13
Coordinating overlapped operations, (Remote Groups Only) 9-11
Copying Test Scripts 9-12

Current accuracy 15-9
Output 15-9
Current measurement accuracy 15-10, 15-14
Current measurement accuracy limits 15-15, 15-16

D

Data store
Commands 7-4
Front panel 7-2
Overview 7-2
Programming examples 7-9
Remote 7-4
DCL (device clear) 11-6
Default settings 1-21
Default setups 1-22
Differences, remote vs. local operation ..2-43
Digital I/O port 10-2
+5V output 10-3
Bit weighting 10-5, 10-10
Commands 10-5
Configuration 10-2, 10-3
Controlling I/O lines 10-4
Front panel control 10-5
Interlock 10-8
Lines 10-3
Output enable 10-3, 10-7
Output enable control 10-8
Programming examples 10-6
Remote operation 10-5, 10-10
Digits 6-5, 6-6, 7-9, 7-10, 7-11
Remote programming 6-6
Display
Annunciators 1-8
Resolution 6-6
Display modes 1-16
Display operations
Adding menu entries 14-11
Character codes 14-6
Clearing 14-4
Deleting menu entries 14-12
Functions and attributes 14-2
Input prompting 14-7
Keycodes 14-13
Load test menu 14-10
LOCAL lockout 14-10
Measurement functions 14-3
Menu 14-7
Messages 14-4
Resolution 14-3
Running a test 14-12
Text messages 14-5
Triggering 14-12
User screen 14-2
DISPLAY PATTERNS test 17-3
Duty cycle 8-23

E

Editing 1-17
Compliance 1-17
Controls 1-17
Source 1-17
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 11-7
Error and status messages 1-21
Error messages
Effects on scripts B-2
Summary B-2
Examples
Ohms programming example 4-15, 4-17, 4-19
Source-measure programming example 4-10

F

Factory scripts 2-42, 13-2
Firmware upgrade 13-35
Modifying 2-42
Fall time 8-22
Features 1-2
Filter
Front panel control 6-10
Filters 6-10
Commands 6-13
Configuration menu 6-10, 6-11
Configuring 6-10
Enabling 6-11
Front panel control 6-10
Programming example 6-13
Remote programming 6-13
Response time 6-10
Types of 6-10
Floating a SMU 3-20
Flow control (signal handshaking) 11-10
Front panel
GPIB operation 11-7
Tests 17-3
Front panel calibration 16-5
Front panel operation
Source-measure procedure 4-8
Front panel summaries 1-6
Functions 2-5, 2-46, 12-3
Fuse
Line, replacement 17-2
Fuse replacement 1-14

G

GET (group execute trigger) 11-7
GPIB
Connections 11-2
Error and status messages 11-7
Front panel operation 11-7
Operation 11-2
Primary address 11-4
Standards 11-2
Status indicators 11-7
Terminator 11-5
GPIB adapter 1-5
GPIB cables 1-4
GTL (go to local) 11-6
Guard 8-18

Connections 8-20
Overview 8-19
Guarding 3-12

H

Hardware trigger modes 10-15

I

IFC (interface clear) 11-6
Input/output connections 1-10, 1-12
Input/output terminal blocks 3-2
Inspection 1-3
Interactive script 2-35
Interface
Selecting an 11-2
Selection 1-20
Interface selection
GPIB 1-21
RS-232 1-21
Interlock 10-8

K

Keys
Function 1-7
Output control 1-8
Range 1-8
Special 1-7
KEYS test 17-3

L

Line frequency 1-14
Line fuse replacement 17-2
Line power 15-3
Line power connection 1-14
LLO (local lockout) 11-6
LOCAL key 11-8
Logical operators 2-48
Loop control 2-51
Low range limits 6-3
LSTN 11-8

M

Math library functions 2-54
Measure
V or I 8-17
Measure only 4-11
Measure voltage or current 4-11
Menus
Configuration 1-20
Main 1-19
Navigation 1-18

N

Noise shield 3-13
Non-volatile memory 2-7
NPLC caching 4-6

O

Ohms
Calculations 4-12
Measurement procedure 4-12
Measurements 4-12
Programming example 4-15
Remote programming 4-15
Sense selection 4-14
Sensing 4-13
Operating boundaries 8-7
Continuous power 8-8
I-Source 8-9
Source or sink 8-7
V-Source 8-13
Operation
Considerations 4-6
Overview 4-2
Options 1-4
Output current accuracy 15-9
Output current accuracy limits 15-10
Output enable 10-7
Control 10-8
control 10-8
Operation 10-7, 10-8
Output voltage accuracy 15-6
Output voltage accuracy limits 15-8
Output-off states 3-23
Overheating protection 8-4

P

Phone number 1-3
Power
Calculations 4-16, 8-5
Equations 8-4
Measurement procedure 4-16, 4-18
Measurements 4-16
Programming example 4-17
Remote programming 4-16, 4-18
Power module 1-10, 1-12
Power switch 1-7
Power-on setup 1-22
Power-up 1-14
Sequence 1-15
Precedence 2-48
PRESet default setup 1-22
Primary address 11-4
Pulse
Concepts 8-22
Duty cycle 8-23
Period 8-22
Rise and fall times 8-22
Sweeps 5-6
Pulse characteristics
Pulse duty cycle 8-23
Pulse energy limitations 8-21
Pulse sweeps 8-4

Q

Queries 2-4
Queues D-2, D-27
Error D-28
Output D-27

R

Range 6-2
Auto 6-3
Auto range limits 6-3
Available ranges 6-2

Commands 6-4
Considerations 6-4
Limitations 6-3
Low limits 6-3
Manual 6-3
Programming 6-4
Programming example 6-5
Reading buffers 12-5
Readings
Maximum 6-3
Recalling 7-3
Requesting 1-24, 4-10
Storing 7-3
Rear panel summaries 1-9
Recalling readings 7-3
Recommended test equipment 15-3
Recommended verification equipment 15-3
Registers
Enable and transition D-13
Measurement event D-24
Operation event D-18
Programming example D-27
Questionable event D-21
Reading D-10
Serial polling and SRQ D-12
Service request enable D-12
Standard event D-16
Status sets D-15
System summary event D-15
Rel 6-8
Defining a value 6-8
Enabling and disabling 6-8
Front panel 6-8
Remote programming 6-9
REM 11-7
Remote calibration 16-5
Remote operation
Source-measure procedure 4-9
Remote programming 1-24
Ohms 4-15
Power 4-16, 4-18
Remote TSP-Link synchroniaztion
line commands 10-10
Removing Stale Buffers 9-12
REN (remote enable) 11-6
Restoring factory defaults 15-4
Rise time 8-22
Rotary knob 1-8
RS-232 cable 1-5
RS-232 Interface
Terminator 11-9
RS-232 interface
Baud rate 11-9
Connections 11-11
Data bits 11-10
Flow control 11-10
Operation 11-8
Parity 11-10
Sending and receiving data 11-9
Run-time environment 2-3, 2-7
Memory considerations 2-44

S

Safety shield 3-16
Safety symbols and terms 1-3
Script management 2-40
Scripting 2-3
Scripting Language 2-3
Scripts 2-4, 2-6
Creating 2-15
Creating functions 2-5
Factory scripts 2-42
Importing 2-24
Launch configuration 2-19
Launching 2-22
Modifying 2-15
Programming model 2-9
Retrieving from Model 260x 2-23
Saving 2-16
User 2-34
Scripts, named 2-4
SDC (selective device clear) 11-7
Selecting an interface 11-2
Sensing 3-5
2-wire local 3-6
4-wire remote 3-7
Mode selection 3-8
Ohms 4-13
Serial polling D-12
Setting the measurement range 15-6
Setting the source range and
output value 15-5
Settling time considerations 8-23
Setups
Front panel 1-21
Power-on 1-22
Restoring 1-21
User 1-21
Shielding
Noise 3-13
Safety 3-16
Sink 8-7
Sink operation 4-12
SMU connections 3-10
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 4-8
Programming example 4-10
Remote operation 4-9
SPE, SPD (serial polling) 11-7
Specifications A-1
Speed 6-6
Command 6-7
Configuration menu 6-7
Programming example 6-7
Remote programming 6-7
Setting 6-7
SRQ (Service Request) 11-8, D-10
Staircase sweeps 8-3
Standard libraries 2-53
Status byte and service request (SRQ) D-10
Status byte and service request
commands D-12

Status model
Clearing registers and queues D-8
Commands D-8
Programming registers and queues D-9
Status byte and SRQ D-2, D-10
TSP-Link system D-28
Status register sets D-2, D-15
Storing readings 7-3
String library functions 2-54
Sweep
Characteristics 5-3
Custom 5-6
Custom functions 5-9
Functions 5-7
Linear staircase 5-3
List 5-6
Logarithmic staircase 5-4
Measurement storage 5-7
Overview 5-2
Programming examples 5-9
Pulse 5-6, 8-4
Staircase functions 5-8
Waveforms 8-3
Syntax rules 12-4
System connections 2-9

T

Tables/arrays 2-47
TALK 11-7
Terminator 11-5, 11-9
Test considerations 15-5
Test fixture 3-19
Test Script Builder 2-11
Instrument Console 2-24
Opening communications 2-13
Starting 2-12
Test Script Builder software 2-9
Test Script Processor 2-3
Tests
Front panel 17-3
Timestamp 7-3
Triggering
Front panel 10-13
Measurement 10-12
Remote 10-14
Types of 10-12
TSP advanced features 9-6
TSP-Link
Abort 9-5
Accessing nodes 9-5
Connections 9-2
Initialization 9-3
Master 9-2
Node numbers 9-3
PC-based system 2-43, 9-2
Reset 9-4
reset() command 9-5
Slaves 9-2
Stand-alone system 2-43, 9-2
TSP-Link synchronizations lines 10-10

U

Unpacking 1-3
User script 2-34
Creating 2-36
Modifying 2-40
Running 2-38
Saving 2-37
User setups 1-21, 1-22
Using the Data Queue 9-11

V

Variables 2-45
Vents
Cooling 1-10, 1-12
Verification 15-2
Limits 15-3
Test procedures 15-5
Test requirements 15-2
Test summary 15-5
Voltage (measure) 4-11
Voltage accuracy 15-6
limit 15-9
Voltage measurement accuracy 15-8
Voltage measurement accuracy limits ... 15-9

W

Warm-up 4-6
Warm-up period 15-2
Warranty information 1-3

This page left blank intentionally.

KEITHLEY

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
❑ Will not read applied input
❑ Overload

❑ Calibration only
❑ Certificate of calibration required
❑ Data required

(attach any additional sheets as necessary)

Show a block diagram of your measurement system 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 GREATER MEASURE OF CONFIDENCE

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY • www.keithley.com

12/06