

Series 3700 System Switch/Multimeter

Reference Manual

3700S-901-01 Rev. C / July 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

Series 3700

System Switch/Multimeter

Reference Manual

©2008, Keithley Instruments, Inc.

Cleveland, Ohio, U.S.A.

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.

Document Number: 3700S-901-01 Rev. C / July 2008

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

Introduction	1-1
Contact information	1-1
Overview	1-1
Measure and switching capabilities	1-2
Warranty information	1-2
Displaying the unit's serial number	1-3
TSP Programming Fundamentals	2-1
Introduction	2-1
Test Script Processor (TSPTM)	2-2
Run-time environment	2-3
Queries	2-3
Scripts	2-4
Named scripts	2-5
Programming overview	2-6
Chunk defined	2-6
Script defined	2-7
Run-time environment	2-8
Nonvolatile memory	2-8
TSPTM programming levels	2-8
Programming model for scripts	2-9
Installing the TSPTM software	2-10
System connections	2-10
Rear panel summary	2-10
GPIB interface connection	2-11
Standard RJ-45 (Ethernet) interface connection	2-12
USB connection	2-13
Using Test Script Builder (TSB)	2-13
Project Navigator	2-14
Script Editor	2-14
Programming interaction	2-15
Sending commands and statements	2-15
Measure voltage	2-15
Read and write to the digital I/O port	2-16
Display user-defined messages	2-16
User scripts	2-16
Script examples	2-17
Creating a user script	2-20
Saving a user script	2-22
Loading a user script	2-24
Running a user script	2-25

Loading a script from the Series 3700 front panel.....	2-28
Saving a script from the Series 3700 front panel	2-29
Modifying a user script	2-29
Script management	2-30
Differences: Remote versus local state	2-32
Remote state.....	2-32
Local state.....	2-32
TSP-Link™ system.....	2-32
Test Script Language (TSL) Reference.....	2-33
Introduction	2-33
Variables and types	2-34
Operators	2-34
Functions	2-35
Tables/arrays	2-36
Precedence.....	2-37
Logical operators	2-38
Concatenation.....	2-39
Branching.....	2-40
Loop control	2-41
Standard libraries.....	2-42
TSP Advanced Features	3-1
Introduction.....	3-1
Using groups to manage nodes on TSP-Link™ network.....	3-4
Master node overview	3-5
Group leader overview	3-5
Assigning groups	3-5
Reassigning groups	3-6
Running parallel test scripts	3-6
Coordinating overlapped operations in remote groups	3-7
Using the data queue for real-time communication.....	3-8
Copying test scripts across the TSP-Link™ network.....	3-8
Removing stale values from the reading buffer.....	3-9
Commands related to TSP advanced features	3-10
Using the Front Panel	4-1
Front panel introduction.....	4-1
Display	4-4
Channel type indication.....	4-8
Using the front panel with non-switch channels	4-9
Special keys and power switch	4-11
CONFIG key	4-11
CONFIG CHAN key	4-11
DISPLAY key	4-16

POWER switch	4-17
RESET switch	4-17
Operation keys	4-17
CHAN key	4-17
DELETE key	4-20
DMM key	4-21
ENTER key	4-25
EXIT key	4-25
FILTER key	4-25
FUNction key	4-26
INSERT key	4-26
LIMIT key	4-27
LOAD key	4-27
MENU key	4-28
PATT key	4-29
REL key	4-30
RUN key	4-30
SCAN key	4-31
SLOT key	4-32
TRIG key	4-32
Range keys, cursor keys, and navigation wheel	4-32
AUTO key	4-32
CURSOR keys	4-32
Navigation wheel	4-33
RANGE keys	4-33
Action keys	4-33
CLOSE key	4-33
OPEN ALL key	4-33
OPEN key	4-34
RATE key	4-34
RECall key	4-34
STEP key	4-34
STORE key	4-35
Range, Digits, Rate, Bandwidth, and Filter	5-1
Range	5-1
Measurement ranges and maximum readings	5-1
Manual range keys	5-2
Auto ranging over the front panel	5-3
Scanning	5-3
Range remote programming (ICL)	5-3
Digits ICL programming	5-4
Scanning	5-4
Setting digits	5-4
Rate	5-5
Setting Rate from the front panel	5-7
Setting measurement speed from a remote interface	5-7
Bandwidth	5-7

Filter.....	5-8
Filter characteristics	5-8
Digital filter window	5-10
Relative, Math, and dB.....	6-1
Relative.....	6-1
Basic front panel REL procedure	6-2
REL remote operation.....	6-2
Scanning.....	6-3
Math calculations.....	6-3
mX+b	6-4
Percent.....	6-6
Reciprocal (1/X)	6-7
dB commands.....	6-10
dB configuration	6-10
dB scanning	6-11
Buffer: Data Storage and Retrieval.....	7-1
Buffer overview.....	7-1
Front panel operation	7-2
Creating and selecting a reading buffer	7-2
Selecting a reading buffer	7-3
Storing readings.....	7-3
Saving readings	7-3
Clearing readings.....	7-4
Deleting a reading buffer.....	7-5
Recalling readings	7-5
Buffer configuration (front panel).....	7-6
Appending readings	7-7
Remote buffer operation.....	7-7
Data store (buffer) commands	7-8
Reading buffers.....	7-12
Time and date values.....	7-16
Buffer status.....	7-16
Dynamically-allocated buffers	7-17
Dynamic buffer programming example	7-18
Buffer for...do loops.....	7-19
Exceeding reading buffer capacity	7-21
Scanning.....	8-1
Scanning fundamentals.....	8-1
Channel assignments.....	8-2
Events	8-2
Foreground and background scan execution	8-3
Trigger model.....	8-4

Trigger model components	8-5
Scan and step counts	8-7
Basic scan procedure	8-7
Buffer	8-9
Changing channel and DMM attributes of an existing scan	8-9
Front panel scanning	8-10
Scan configuration	8-11
Bus operation scanning	8-12
ICL commands	8-12
Scanning examples	8-14
Hardware trigger modes	8-18
Falling edge trigger mode	8-20
Rising edge master trigger mode (version 1.4.0 or higher)	8-21
Rising edge acceptor trigger mode (version 1.4.0 or higher)	8-22
Either edge trigger mode	8-23
Understanding synchronous triggering modes	8-24
Files	9-1
File formats	9-1
Default file extensions	9-1
File system navigation	9-2
File I/O	9-3
Script examples	9-4
Command table entries	9-9
TSP-Net	10-1
Overview	10-1
TSP-Net™ Capabilities	10-1
Using TSP-Net™ with any Ethernet-enabled device	10-2
Example script	10-3
Using TSP-Net™ vs. TSP-Link™ for communication with TSP-enabled devices	10-4
Instrument Control Library (ICL) - General device control	10-5
Instrument Control Library - TSP-specific device control	10-12
LXI Class B Triggering (IEEE-1588)	11-1
Introduction to IEEE-1588 based triggering	11-1
IEEE-1588 implementation in the Series 3700	11-1
Correlating PTP to Coordinated Universal Time (UTC)	11-2

Configuring and enabling IEEE-1588	11-3
Scheduling alarms	11-5
Monitoring alarms with LAN triggers and LXI event log	11-6
LXI event log	11-7
Example applications of IEEE-1588 in Series 3700-based systems	11-7
Synchronizing multiple Series 3700 instruments	11-9
Status Model	12-1
Status register sets	12-1
Negative and positive transition registers	12-2
Status byte and SRQ	12-2
Queues	12-2
System summary and status byte	12-3
System summary registers	12-4
Standard event status register and enable	12-5
Operation events registers	12-6
Questionable event register	12-7
Measurement event register (measurement)	12-8
Status function summary	12-8
Clearing registers and queues	12-9
Programming enable and transition registers	12-10
Reading registers	12-11
Status byte and service request (SRQ)	12-12
Status byte register	12-13
Serial polling and SRQ	12-14
Service request enable register	12-14
SPE, SPD (serial polling)	12-14
Status byte and service request commands	12-15
Enable and transition registers	12-15
Controlling node and SRQ enable registers	12-16
Status register set specifics	12-16
System summary event registers	12-16
Standard event register	12-19
Operation event registers	12-21
Questionable event registers	12-23
Measurement event registers	12-24
Queues	12-25
Output queue	12-25
Error queue	12-26

Instrument Control Library (ICL)	13-1
Command programming notes	13-1
Wild characters	13-1
Functions and attributes	13-2
TSP-Link TM nodes	13-5
Logical instruments	13-5
Query commands	13-6
DMM configuration	13-8
ICL command list	13-11
beeper functions and attributes	13-16
bit functions	13-17
channel functions and attributes	13-24
dataqueue functions and attributes	13-85
delay functions	13-86
digio functions and attributes	13-87
display functions and attributes	13-93
dmm functions and attributes	13-109
errorqueue functions and attributes	13-176
eventlog functions and attributes	13-177
exit functions	13-180
file functions	13-181
format attributes	13-183
fs functions	13-186
gpib attributes	13-187
io functions	13-188
LAN functions and attributes	13-190
localnode functions and attributes	13-210
makegetter functions	13-218
memory functions	13-219
opc functions	13-220
print functions	13-221
ptp functions and attributes	13-223
reset functions	13-230
scan functions and attributes	13-230
schedule functions and attributes	13-250
setup functions and attributes	13-252
slot[X] attributes	13-255
status functions and attributes	13-264
timer functions	13-286
trigger functions and attributes	13-287
trigger.timer functions and attributes	13-290
tslink functions and attributes	13-294
tslink.trigger functions and attributes	13-295
tspnet functions and attributes	13-300
upgrade functions	13-309
userstring functions	13-310
waitcomplete functions	13-312

Verification.....	14-1
Introduction.....	14-1
Verification test requirements.....	14-2
Environmental conditions.....	14-2
Warmup period.....	14-2
Line power.....	14-3
Recommended test equipment.....	14-3
Verification limits.....	14-4
Restoring factory defaults.....	14-5
Performing the verification test procedures.....	14-5
Test summary.....	14-5
Test considerations.....	14-6
Series 3700 verification tests.....	14-6
Verifying DC voltage.....	14-6
Verifying AC voltage.....	14-9
Verifying DC current 10 μ A to 100 μ A ranges.....	14-11
Verifying DC current 1mA to 3A ranges.....	14-13
Verifying AC current 1mA to 3A ranges.....	14-15
Verifying frequency.....	14-18
Verifying 4-wire resistance.....	14-19
Verifying 2-wire resistance.....	14-21
Verifying dry circuit resistance.....	14-22
Verifying 1-OHM and 10-OHM resistance ranges.....	14-24
Verifying zeros using a 4-wire short.....	14-25
Calibration.....	15-1
Overview.....	15-1
Environmental conditions.....	15-2
Warmup period.....	15-2
Line power.....	15-2
Calibration considerations.....	15-3
Calibration cycle.....	15-3
Recommended equipment.....	15-3
Calibration.....	15-4
Remote calibration procedure.....	15-5
DC volts calibration.....	15-6
Resistance calibration.....	15-8
DC current calibration.....	15-9
AC volts calibration.....	15-11
AC current calibration.....	15-13
Frequency calibration.....	15-15
Save calibration.....	15-16

Maintenance	16-1
Introduction.....	16-1
Fuse replacement.....	16-1
Front panel tests.....	16-3
Test procedure.....	16-3
Error and status messages	17-1
Introduction.....	17-1
Error summary.....	17-1
Error effects on scripts.....	17-1
Reading errors.....	17-2
Error and status message list.....	17-2
Appendix A: EEE-1588 Glossary of Terms.....	A-1
Boundary clock.....	A-1
Epoch.....	A-1
Grandmaster clock	A-1
Master clock	A-2
PTP.....	A-2
PTP port	A-2
PTP subdomain	A-2
Index.....	I-1

List of Figures

Figure 1-1: DMM measurement capabilities	1-2
Figure 2-1: TSP test script example	2-7
Figure 2-2: Programming model for scripts	2-9
Figure 2-3: Rear panel features	2-10
Figure 2-4: GPIB cable	2-11
Figure 2-5: Using Test Script Builder (TSB)	2-14
Figure 3-1: Multiple TSP-Link networks.....	3-2
Figure 3-2: Single TSP-Link network with groups	3-3
Figure 4-1: Model 3706 System Switch/Multimeter	4-2
Figure 4-2: Model 3706-S System Switch (no DMM)	4-2
Figure 4-3: Model 3706-NFP System Switch/Multimeter.....	4-3
Figure 4-4: Model 3706-SNFP System Switch (no DMM)	4-3
Figure 4-5: Active channel display example	4-5
Figure 4-6: MAIN MENU display.....	4-7
Figure 5-1: Speed versus noise characteristics.....	5-5
Figure 5-2: Moving average filter	5-9
Figure 5-3: Repeating average filter	5-9
Figure 5-4: Filter window.....	5-11
Figure 8-1: Event detector	8-2
Figure 8-2: Trigger model	8-4
Figure 8-3: Falling edge input trigger	8-20
Figure 8-4: Falling edge output trigger.....	8-20
Figure 8-5: RisingM output trigger	8-21
Figure 8-6: RisingA input trigger	8-22
Figure 8-7: RisingA output trigger.....	8-22

Figure 8-8: Either edge input trigger	8-23
Figure 8-9: Either edge output trigger	8-24
Figure 8-10: SynchronousM input trigger	8-25
Figure 8-11: SynchronousM output trigger	8-26
Figure 8-12: SynchronousA input trigger	8-27
Figure 8-13: SynchronousA output trigger	8-27
Figure 8-14: Synchronous input trigger	8-28
Figure 8-15: Synchronous output trigger	8-29
Figure 12-1: Status byte and queues	12-2
Figure 12-2: Status byte and system summary register	12-3
Figure 12-3: System summary registers	12-4
Figure 12-4: Standard event registers and event status enable	12-5
Figure 12-5: Operation event registers	12-6
Figure 12-6: Questionable event register	12-7
Figure 12-7: Measurement event register	12-8
Figure 12-8: 16-bit status register	12-10
Figure 12-9: Status byte and service request (SRQ)	12-12
Figure 12-10: Standard event register	12-20
Figure 13-1: ch_list legend	13-24
Figure 13-2: Multiplexer card display	13-26
Figure 13-3: Matrix card display	13-27
Figure 13-4: Status byte and queues	13-264
Figure 14-1: DC voltage verification	14-7
Figure 14-2: AC voltage verification	14-10
Figure 14-3: DC current verification 10 μ A to 100 μ A ranges	14-12
Figure 14-4: DC current verification 1mA to 3A ranges	14-13

Figure 14-5: DC current verification 3A range diagram	14-14
Figure 14-6: AC current verification 1mA to 1A range	14-16
Figure 14-7: AC current verification 3A range	14-16
Figure 14-8: Frequency verification	14-18
Figure 14-9: Resistance verification	14-19
Figure 14-10: 2-wire resistance verification	14-21
Figure 14-11: Resistance verification	14-23
Figure 14-12: Verifying discrete resistance	14-24
Figure 14-13: 4-wire short diagram.....	14-26
Figure 15-1: 4-wire short diagram.....	15-6
Figure 15-2: DC voltage calibration	15-7
Figure 15-3: Resistance calibration	15-8
Figure 15-4: DC current calibration.....	15-9
Figure 15-5: AC voltage calibration	15-11
Figure 15-6: AC current calibration 1mA to 1A range.....	15-13
Figure 15-7: Low frequency calibration.....	15-15
Figure 15-8: Frequency verification	15-15
Figure 16-1: Fuse location	16-2

Introduction

In this section:

Contact information	1-1
Overview	1-1
Warranty information	1-2

Contact information

If you have any questions after reviewing this information, please contact your local Keithley Instruments representative or call one of our Applications Engineers at 1-888-KEITHLEY (1-888-534-8453). You can also contact us through our [website](http://www.keithley.com) (<http://www.keithley.com>).

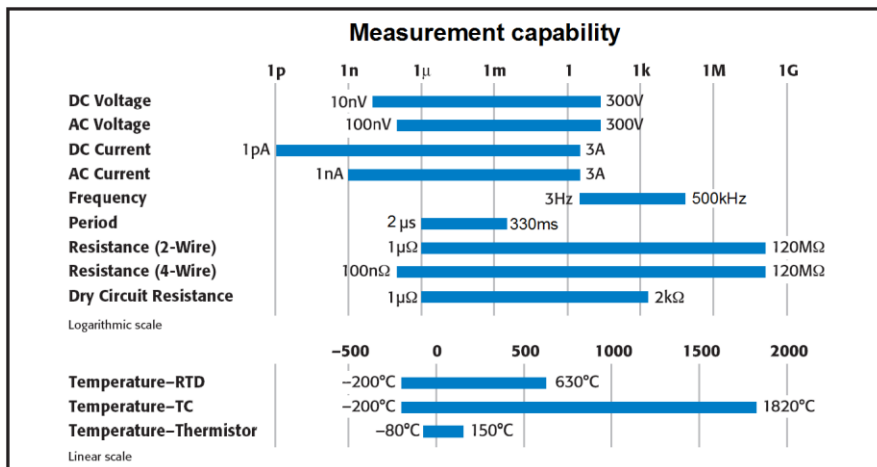
Overview

The Keithley Instruments Series 3700 System Switch/Multimeter features scalable, instrument grade switching and multi-channel measurement solutions that are optimized for automated testing of electronic products and components. The Series 3700 includes four versions of the Model 3706 system switch mainframe, along with a growing family of plug-in switch and control cards. When the Model 3706 mainframe is ordered with the high performance multimeter, you receive a tightly-integrated switch and measurement system that can meet the demanding application requirements in a functional test system or provide the flexibility needed in stand-alone data acquisition and measurement applications.

Measure and switching capabilities

The basic measurement capabilities of Series 3700 systems are summarized in the following figure.

Figure 1-1: DMM measurement capabilities



Warranty information

Detailed warranty information is located at the front of this manual. Should your Series 3700 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 the service form at the back of this manual and give it to the repair facility with all relevant information.

NOTE The service form requires the serial number of the Series 3700. The serial number label is located inside the unit on the bottom panel. The serial number can be viewed by removing the slot covers and/or switching modules from the mainframe.

WARNING *Before removing (or installing) switching modules, make sure you turn off the Series 3700 and disconnect the line cord. Also, remove any other external power connected to the instrument or switching module(s).*

Failure to disconnect power before removing (or installing) switching modules may result in personal injury or death due to electric shock.

Displaying the unit's serial number

To display the serial number on the front panel:

NOTE If the Series 3700 is in remote mode, press the **EXIT** key once to place the unit in local mode.

1. When in local mode, press the **MENU** key.
2. Scroll to the **SYSTEM-INFO** menu and press the **ENTER** key.
3. On the **SYSTEM INFORMATION** menu, scroll to the **SERIAL#** and press the **ENTER** key. The Series 3700 serial number will be displayed.

TSP Programming Fundamentals

In this section:

Introduction	2-1
Test Script Processor (TSPTM)	2-2
Run-time environment.....	2-3
Queries	2-3
Scripts	2-4
Named scripts	2-5
Programming overview	2-6
Installing the TSPTM software	2-10
System connections	2-10
Using Test Script Builder (TSB)	2-13
Sending commands and statements	2-15
Measure voltage	2-15
User scripts	2-16
Differences: Remote versus local state.....	2-32
Test Script Language (TSL) Reference.....	2-33

Introduction

Conventional electronic test and measurement equipment responds to command messages sent to the instrument. Each command message contains one or more commands that the instrument executes in order. To conduct a test, a computer controller is programmed to send a sequence of commands to an instrument. The controller orchestrates the actions of the instrumentation. Typically, the controller is programmed to request measurement results from the instrumentation and make test sequence decisions based on those measurements.

In addition to operating as conventional instruments, Keithley Instruments' Test Script Processor (TSP)-based instruments are capable of executing scripts that process commands in the instrument rather than needing to be sent from a computer. Basically, a script allows you to have a program running inside the instrument to execute a sequence of commands without the need to send them individually from a computer. Once a script is loaded into the instrument, it only needs to be called (similar to a function) to execute the desired command sequence. In the sections that follow, you will learn what a script is, and how to create, save, and load a script.

Test Script Processor (TSPTM)

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](http://www.lua.org) (<http://www.lua.org>). This makes the instrument fully capable of interpreting and executing code in a way that is similar to Visual Basic (VB) or Java, rather than only responding to single-line commands. Program statements control script execution and provide capabilities such as variables, functions, branching, and loop control.

Because scripts are programs, they are written using a programming language, called the test script language or TSL. TSL is derived from the Lua scripting language. For details about TSL, see the [Test Script Language \(TSL\) reference](#) (on page 2-33).

In this manual, we refer to Lua as the "test script language" or "TSL." The TSP runs portions of TSL code called "chunks." Most messages sent to the instrument are directly executed by the TSP as TSL chunks. The simplest messages sent to the instrument are individual instrument control commands. Even though these messages are executed as TSL chunks, using them is no different than using a conventional instrument. You send 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.

The command set for each TSP-enabled instrument is referred to as the "instrument control library" or ICL. Each TSP-enabled instrument will have its own set of ICL commands. Although each TSP-enabled instrument inherits the same TSL, different instruments extend the language in their own way.

ICL commands are similar to the commands sent to a conventional instrument, but ICL commands appear like function calls or assignment statements. For example, the command to set ASCII precision to 10 for ASCII readings is:

```
format.asciiprecision = 10
```

Similarly, the command to format readings as ASCII is:

```
format.data = format.ASCII
```

These commands do not need to be sent as separate messages. They can be combined into one message by joining the two commands together with a space separating them. The resulting chunk would be as follows:

```
format.asciiprecision = 10 format.data = format.ASCII
```

Run-time environment

A feature of all scripting environments is the run-time environment. In the TSP™, the run-time environment is simply a temporary 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. To instruct the instrument to read the ASCII precision setting and store the result in a global variable named "x", send:

```
x = format.asciiprecision
```

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.

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 creates one response message.

The following chunk is an example that generates an output response message:

```
x = 10 print(x) → 1.000000000e+001
```

NOTE The output (indicated by the →) may vary, depending on the ASCII precision setting.

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. Use the `loadscript` and `endscript` messages to collect a sequence of commands into one chunk.

The `loadscript` message marks the beginning of a script. The instrument will collect all following messages until the `endscript` message is received (the `endscript` message marks the end of the script). The TSP-enabled instrument compiles the test sequence and makes it available to run in a subsequent message. This chunk is called the "anonymous script."

NOTE Every time the `script.run()` command is given, the anonymous script will be executed.

The anonymous script can be run at any time by sending the command `script.run()` or `script.anonymous()`. The anonymous script can be run many times (it remains in active memory until a new anonymous script is created). Sending a new script using the `loadscript` and `endscript` messages will instruct the TSP-enabled instrument to replace the anonymous script with the new script. To see the current contents of the anonymous script, send the command `script.anonymous.list()`.

Creating and using scripts this way is a very powerful feature of TSP-enabled instruments, but it is limited to accessing only one script at a time. The solution to this limitation is to create user-defined named scripts. See [Named scripts](#) (on page 2-5) for information on how to use named scripts, and also how to store many scripts in the instrument at one time.

Named scripts

The `loadscript` message can also be used to create named scripts. Loading a named script does not replace the anonymous script. Instead, a global variable in the run-time environment is temporarily 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. Specify the name of the script in the `loadscript` message by separating it from the `loadscript` keyword with a space character.

For example, 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, you can run it at any time by sending either the `MyScript()` or `script.user.scripts.MyScript()` command.

Named script key points:

- Create different script names using `loadscript`.
- Sending a new script with the same name will overwrite (replace) the previous version.
- Sending new scripts with different names will not remove previously sent scripts.
- 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. Therefore, like all other global variables, they are lost when the unit is powered off.
- Nonvolatile storage can be used to store downloaded scripts across power cycles. See [Saving a user script](#) (on page 2-22) for more information.

Programming overview

Chunk defined

A chunk is a single programming statement or a sequence of statements that are executed sequentially (that is, sent to Lua as a single line). There are non-scripted 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 contain multiple statements. Each statement in the line of code must be separated by white space. 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

Each of the following lines of code is a separate chunk. The first chunk sets the ASCII precision to 10 for readings. The second chunk turns on ASCII readings.

```
format.asciiprecision = 10  
format.data = format.ASCII
```

Scripted chunk

In a script environment, the chunk is the entire listing of test programming code. If the two statements in [Multiple chunks](#) (on page 2-6) were created as a script, then those two lines of code would be assembled as one chunk. The instrument internally constructs a chunk out of a series of messages sent between `loadscript` and `endscript`. Also see [Script defined](#) (on page 2-7) for more details.

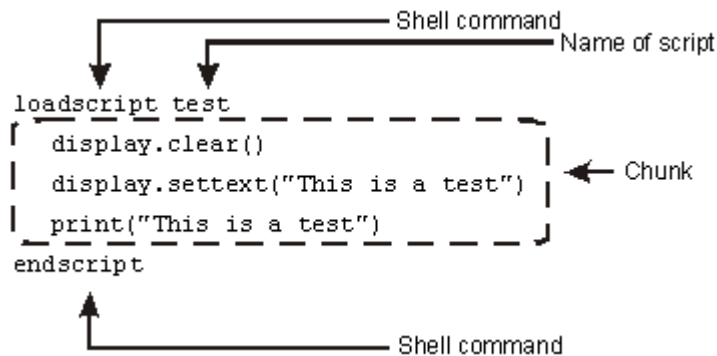
Script defined

The Series 3700 utilizes a Test Script Processor (TSP) to process and run individual chunks or scripts. A script is a collection of instrument control commands and programming statements. The [TSP test script example](#) (on page 2-7) shows an example of how to create and load a script. When this script (named "test") is run, the message "This is a test" will be displayed on the Series 3700 and sent to the computer.

As shown, a script consists of a chunk of programming code framed by shell commands. The first shell command in the TSP script example loads the script named "test." The last shell command marks the end of the script. The chunk in the TSP script example consists of three lines of code. When the chunk is executed, the test messages are sent and displayed. The following command executes the chunk `test()`.

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

Figure 2-1: TSP test script example



A script is loaded into the Series 3700, where it can be run. Running a script using this method is faster than running a test program from the control computer because it eliminates the piecemeal transmission process from the control computer.

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 \(TSB\)](#) (on page 2-13)). The user script is loaded into the Series 3700 and can be saved in nonvolatile memory. These are the scripts referenced as a "user script" throughout the manual.

Run-time environment

The run-time environment is a collection of global variables (scripts) that you have 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 nonvolatile memory of the Series 3700 are automatically recalled into the run-time environment when the instrument is turned on.
- Named scripts that you have created and loaded are also placed in the run-time environment. A named script resides in volatile memory and must be saved to nonvolatile memory to retain it after power-off.
- An unnamed script that you have created and loaded is also placed in the run-time environment. Remember that only one unnamed script, referred to as the "anonymous script," can be in the run-time environment. If another unnamed script is created and loaded, it will replace the old unnamed script in the run-time environment. An unnamed script resides in volatile memory and must be saved to nonvolatile memory to retain it after power-off.

Nonvolatile memory

New or modified user scripts loaded into the Series 3700 reside in the run-time environment and are lost when the unit is turned off. To save a script after power-down, you must save it in nonvolatile memory. When the Series 3700 is turned back on, all saved scripts will load into the run-time environment.

NOTE Do not confuse the run-time environment with the nonvolatile memory of the Series 3700. 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 nonvolatile memory.

TSPTM programming levels

Instrument control library (ICL) commands and Test Script Language (TSL) programming statements are used to program and control the Series 3700. There are two levels of programming:

- *Sending commands and statements* (on page 2-15): Non-scripted chunks are executed one line at a time by the PC.
- *User scripts* (on page 2-16): A program script is run after you have created and loaded it into the Series 3700. An interactive script is a type of script that interacts with the operator. It provides user-defined messages on the Series 3700 display to prompt the operator to enter parameters from the front panel.

Programming model for scripts

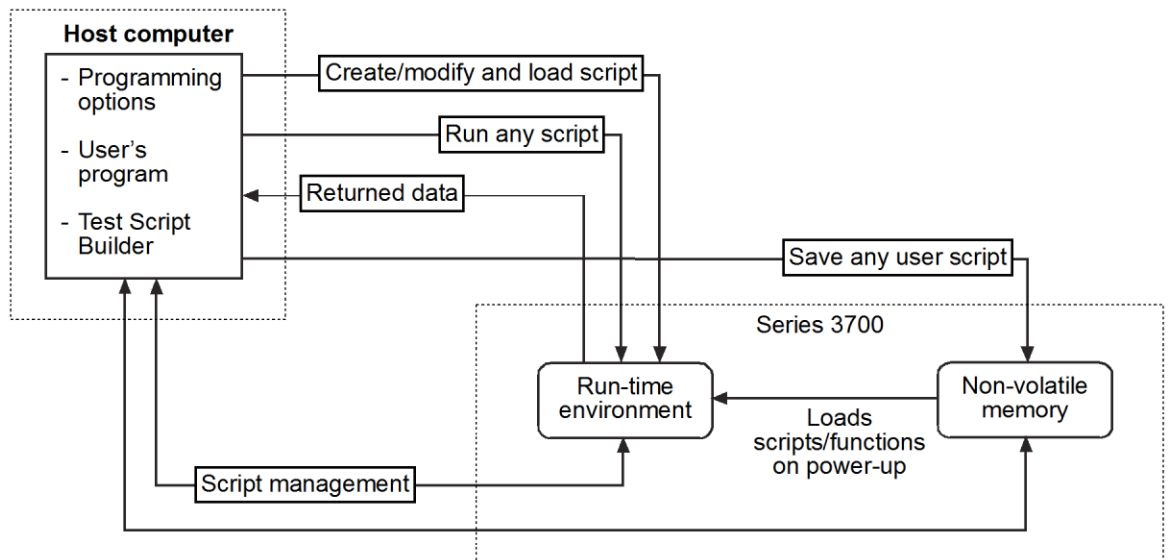
User-created scripts can be stored in nonvolatile memory. When the Series 3700 is turned on, all user script functions are recalled into the run-time environment from nonvolatile memory. If any user scripts have been programmed to run automatically, they will run after all the scripts are loaded. Any script in the run-time environment can be run from the Test Script Builder or the user's own program. Test data (for example, a reading) is returned from the Series 3700 to the computer. 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. Scripts that are not saved are lost when the Series 3700 is turned off.

Script management includes commands for the following operations:

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

The fundamental programming model for scripts is shown in the following figure.

Figure 2-2: Programming model for scripts



Installing the TSPTM software

To install the TSP software:

1. Close all programs.
2. Place the CD (Keithley Instruments part number KTS-850B01 or greater) into your CD-ROM drive.
3. Follow the on-screen instructions.

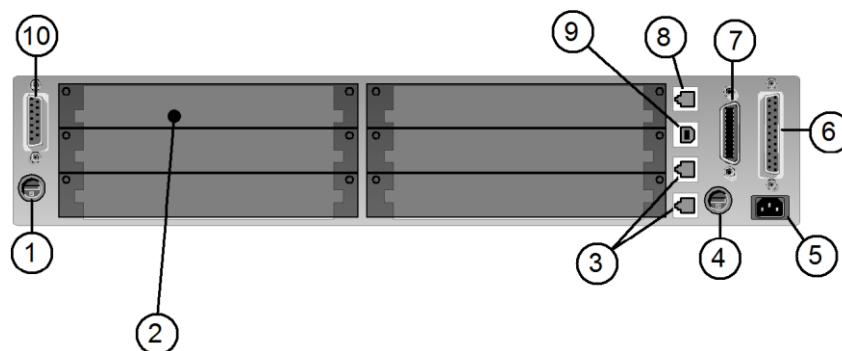
If your web browser does not start automatically and display a screen with software installation links, open the installation file (index.html) found on the CD to initiate automatic installation.

System connections

Up to 64 TSP™ instruments can be used in a test system. The host interface for the test system can be the GPIB, Ethernet, or USB. For the GPIB, an IEEE-488 cable is used to connect the computer to one of the Series 3700 instruments. USB and Ethernet also require the appropriate cables. Note that only one cable is needed to connect to one of the Series 3700 instruments because communication to the other Series 3700 instruments or TSP-enabled products can be accomplished using the TSP-Link™.

Rear panel summary

Figure 2-3: Rear panel features



Item	Description
1	Analog backplane fuse
2	Slots (6 places)
3	TSP-Link™ connectors (2 places)

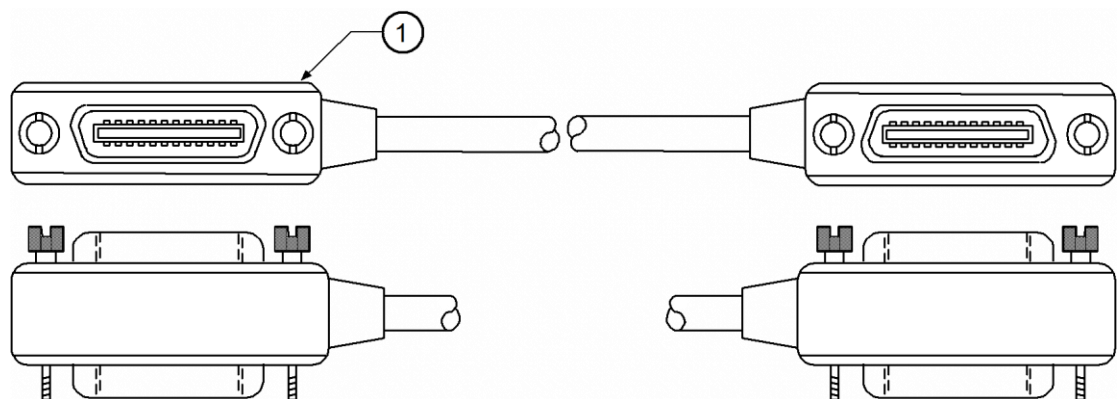
Item	Description
4	Instrument fuse
5	Power connector
6	Digital I/O port
7	GPIB connector
8	Ethernet connector
9	USB connector
10	Analog backplane connector

GPIB interface connection

Use a shielded IEEE-488 cable to connect the Series 3700 IEEE-488 connector to the GPIB connector on the control computer. Connect one end of the cable to the host computer and the other end to Series 3700. Both cable connections (see item 1 in [GPIB cable](#) (on page 2-11)) are identical. The GPIB cable connectors are stackable. For additional non-Series 3700 GPIB instruments in the test system, daisy-chain a GPIB cable from one instrument to another.

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.

Figure 2-4: GPIB cable



Refer to [Rear panel summary](#) (on page 2-10) for connector location.

GPIB address

At the factory, the GPIB is set to address value 16. 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.

Change the GPIB address from the GPIB menu. To access the menu, press the **MENU** key and select **GPIB**. The GPIB address is saved in nonvolatile memory. The address value will not change when power is cycled or a reset command (reset) is sent. For units without a front panel, use the `gpib.address` (on page 13-187) ICL command to change the GPIB address setting remotely.

Standard RJ-45 (Ethernet) interface connection

The Series 3700 uses a standard Ethernet connection configuration. It is designed for a 10/100BaseTX network using standard RJ-45 connectors. This is an eight-wire connector, but only two sets of wire pairs are used: one pair to transmit and one pair to receive data.

A 10BaseT network can accommodate transmission speeds up to 10Mbit per second, where a 100BaseTX network operates at speeds of up to 100Mbit per second. Both types of networks usually require Ethernet hubs to make connections.

The exception is a one-to-one connection using a crossover cable, which may be a 10BaseT or 100BaseTX, depending on the computer's Ethernet interface card and which category of cable is used (the Series 3700 can be directly connected to a computer's NIC card using an Ethernet crossover cable). Refer to [Rear panel summary](#) (on page 2-10) for connector location.

LAN address

Change or view the LAN address from the LAN menu. To access the menu, press the **MENU** key and select **LAN**. The LAN address is saved in nonvolatile memory. The address value will not change when power is cycled or a reset command is sent. For units without a front panel, use the `lan.config` and `lan.status` ICL commands to change or view the LAN settings remotely. See [LAN functions and attributes](#) (on page 13-190) for more information.

USB connection

Connect a computer controller to the Series 3700 rear panel USB (from host) connector (connect USB flash drive devices to the Series 3700 front panel USB). Refer to [Rear panel summary](#) (on page 2-10) for connector location.

NOTE For your Series 3700 to be recognized by your computer over the USB interface, the proper driver must be installed. Installing the Test Script Builder application also installs the applicable USB driver (it becomes available after installing this software). To complete the USB driver installation, after installing the Test Script Builder application, connect the Series 3700 USB connector (rear panel) to the computer.

Using Test Script Builder (TSB)

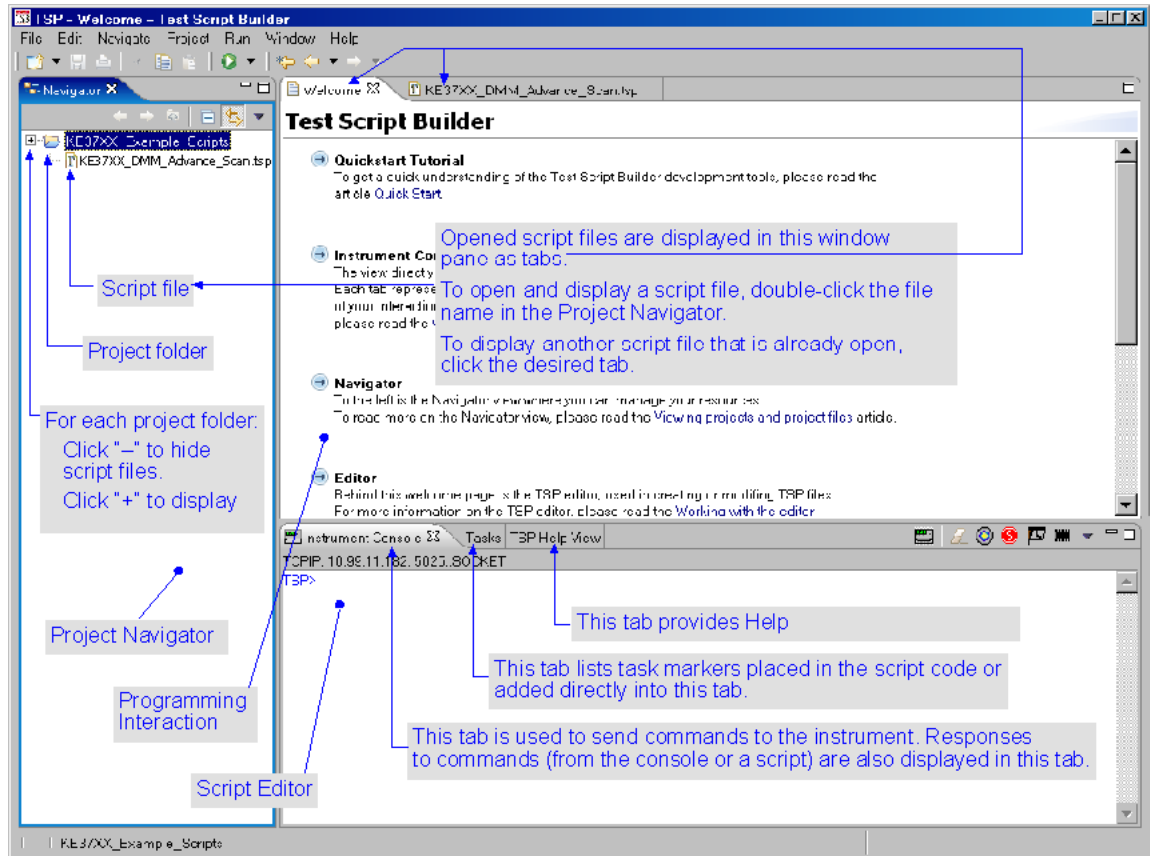
Test Script Builder is a supplied software tool that can be used to perform the following operations:

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

The following figure shows an example of the Test Script Builder. As shown, the Workspace is divided into three window panes:

- [Project Navigator](#) (on page 2-14)
- [Script Editor](#) (on page 2-14)
- [Programming interaction](#) (on page 2-15)

Figure 2-5: Using Test Script Builder (TSB)



Project Navigator

The Project Navigator resides in the window pane on the left side of the workspace. The navigator consists of 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 the graphic in *Using Test Script Builder (TSB)* (on page 2-13) contains a project named KE37XX_Example_Scripts, which has one script file, called KE37XX_DMM_Advance_Scan.tsp.

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 Series 3700, where it can be run.

Programming interaction

Up to seven tabs can be displayed in the lower pane of the workspace (the script editor) to provide programming interaction between the Test Script Builder and the Series 3700. The instrument console shown in [Using Test Script Builder \(TSB\)](#) (on page 2-13) is used to send commands to the connected Series 3700. Retrieved data (for example, readings) from commands and scripts appear in the instrument console.

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 (for example, readings) are then transmitted back to the computer.

Measure voltage

The digital multimeter (DMM) of the Series 3700 is capable of measuring various functions. The following code fragment programs the DMM to measure DC voltage at a specified NPLC and range.

```
Return the Series 3700 to default settings.
reset()

Set the DMM function to DC volts
dmm.func = 'dcvolts'

Set the NPLC for DC volts
dmm.nplc = 0.1

Set the range for DC volts
dmm.range = 10

Take the DC volts measurement
reading = dmm.measure()

Displays measure voltage reading
print(reading)
```

Read and write to the digital I/O port

The digital I/O port of the Series 3700 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). See the pinout in the graphic in [Rear panel summary](#) (on page 2-10) for the Digital I/O port for additional information.

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.

Command	Description
<code>digio.writebit(4,0)</code>	Writes a 0 to bit 4
<code>data=digio.readbit(4)</code>	Reads value at bit 4
<code>print(data)</code>	Displays data on PC

Display user-defined messages

The operator can define and display messages on the front panel display of the Series 3700. The following code fragment displays the "Test in Process" message on the Series 3700 display:

Command	Description
<code>display.clear()</code>	Clears the display
<code>display.settext('Test in Process')</code>	Displays user message

Displayed messages and input prompts are used in scripts to prompt the operator to enter parameter values from the front panel. See [Interactive script](#) (on page 2-19) 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 3700 and can be saved in nonvolatile memory. Scripts not saved in nonvolatile memory will be lost when the Series 3700 is turned off.

Script examples

Script using commands and statements only

The following script closes Channels 1-10 on Slot 3 and measures voltage on each channel. The ten voltage readings are returned to the host computer, as well as being stored in a voltage table on the instrument, using the channel numbers as keys to index the table.

Example exclusive close and measure scripts follow:

User script created in Test Script Builder

```
voltage = { }
reset()
for j = 3001,3010 do
    channel.exclusiveclose('3911,' .. j)
    voltage[j] = dmm.measure()
    print(voltage[j])
end
```

User script created in user's own program

```
loadscript
    voltage = { }
    reset()
    for j = 3001,3010 do
        channel.exclusiveclose('3911,' .. j)
        voltage[j] = dmm.measure()
        print(voltage[j])
    end
endscript
```

NOTE When creating a script using the Test Script Builder, only the chunk is typed in as shown in the Test script builder example. See [Using Test Script Builder \(TSB\)](#) (on page 2-13) for details on creating, loading, and running the script. When creating a script using a programming language (the User's program script example), shell commands must be included to manage interactions between the host computer and TSP™. The `loadscript` command starts loading the script into the Series 3700 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 following script contains an ICL command to reset the DMM and a function (named `MyDcv`), which takes one parameter to represent the speed of the measurement. When this script is run, the DMM will be reset and the function `MyDcv` will be available for calling.

Example scripts using a function:

Test script builder example

```
dmm.reset('all')
function MyDcv(speed)
dmm.func = 'dcvolts'
    dmm.nplc = speed
    dmm.range = 10
    reading = dmm.measure()
    print(reading)
end
```

User's program script example

```
loadscript
    dmm.reset('all')
    function MyDcv(speed)
        dmm.func = 'dcvolts'
        dmm.nplc = speed
        dmm.range = 10
        reading = dmm.measure()
        print(reading)
    end
endscript
```

When calling the function, you must specify the measurement speed in the argument for the function. For example, to set the measurement speed to 0.5, call the function as follows:

```
MyDcv(0.5)
```

This will set the DMM function to DCV, NPLC to 0.5, and range to 10V. The voltage reading is sent to the host computer and displayed.

Interactive script

An interactive script prompts the operator (on the Series 3700 display) to input test parameters (using the Series 3700 front panel). The chunk fragment in the following table uses display messages to prompt the operator to select a measure function (DCV or 2-wire) and a range based on function, and to input the measurement speed. 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 measurement speed. If a value is not entered, the default level (1) will be set when **ENTER** is pressed. The operator will not be able to input values that are not within the limits (minimum of 0.01 and maximum of 3).

Example of an interactive script chunk fragment (Test Script Builder or user's program):

```
-- Prompt operator to select function:
myfunc = display.menu ('Select function', 'dcvolts
    twowireohms')

-- Now prompt for range based on function selected
if (myfunc == 'dcvolts') then
    myrange = display.menu('Select range', '10 100')
    if (myrange == '10') then
        range_value = 10
    else
        range_value = 100
    end
else
    myrange = display.menu('Select range', '1000 10000')
    if (myrange == '1000') then
        range_value = 1000
    else
        range_value = 10000
    end
end

-- Prompt operator to set (input) measurement speed
speed = display.prompt('0.00', 'NPLC', 'Enter measure
    speed', 1, 0.01, 3)

-- Wait for operator to set the measurement speed
dmm.reset('all')
dmm.func = myfunc
dmm.range = range_value
dmm.nplc = speed
print(dmm.measure())
```

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 3700. The script on the left is unnamed (anonymous script), while the one on the right is named (where name is the user-defined name):

<code>loadscript</code>	<code>loadscript name</code>
<code>(chunk)</code>	<code>(chunk)</code>
<code>endscript</code>	<code>endscript</code>

Load and run

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

<code>loadandrunscript</code>	<code>loadandrunscript name</code>
<code>(chunk)</code>	<code>(chunk)</code>
<code>endscript</code>	<code>endscript</code>

Details about `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 you are 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 nameless script (anonymous script) is loaded or when the Series 3700 is turned off. After loading the unnamed script (anonymous), use the `run()`, `script.run()`, `script.anonymous()`, or `script.anonymous.run()` commands to run it.

A script saved in nonvolatile memory named `autoexec` has special properties that cause it to run automatically after the Series 3700 is powered on and all `autorun` scripts have been executed. For details, see [Autoexec script](#) (on page 2-26) and [Autorun scripts](#) (on page 2-26).

```
loadandruncscript  
loadsandruncscript 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 [Autorun scripts](#) (on page 2-26)).

Creating a user script (alternative)

An alternate way to create a script is to use `script.new`, which creates a script from a chunk of Lua code, using the following command:

```
myscript = script.new(code, name)
```

Where:

- `myscript` is the created script or nil, if an error occurred. If the `name` parameter is an empty string, this is the only handle to the created script.
- `code` is the string representing a chunk of Lua code, which will be used as the script body.
- `name` (optional) is the name of the script to be created. The script's `name` attribute is initialized to this value, which (if not the empty string) also serves as the key used to access the script through the `script.user.scripts` table. The default is the empty string.

If the name of the `script.new` script conflicts with the name of an existing script in the `script.user.scripts` table, the existing script will be unnamed (that is, its `name` attribute will be set to the empty string) before it is replaced in the `script.user.scripts` table by the newly-created script.

Example:

To create a new global script called `MyTest8` and assign the `name` attribute to `MyTest8` that displays "Hello from MyTest8" on the display, use the following command:

```
MyTest8 = script.new("display.clear()  
display.settext('Hello from MyTest8')", 'MyTest8')
```

To run this script:

```
MyTest8()
```

Get or change the name attribute of a script

The following commands are used to get or change the name attribute of a script:

```
val = myscript.name -- read the name of myscript
myscript.name = val -- change the name of myscript
```

Where:

- `myscript` is the script
- `val` is the name of the script

This attribute may optionally be initialized when the script is created. See [Loading a user script](#) (on page 2-24) (`script.load`) and [Creating a user script \(alternative\)](#) (on page 2-21) (`script.new`) for details.

This attribute must be either a valid Lua identifier or the empty string. Changes to its value are reflected in the `script.user.scripts` table. Setting the attribute to the empty string will remove the script from the table completely.

For example, a new script can be created without naming it, such as:

```
MyTest7 = script.new("display.clear()
    display.settext('Hello from my test')")
```

To run this script, type:

```
MyTest7()
```

If `print(MyTest7.name)` is executed, it will print a blank line because the name is an empty string.

To name the script "MyTest7" and read the name, send:

```
MyTest7.name = 'MyTest7'
print(MyTest7.name) --> MyTest7 is displayed
```

Saving a user script

A created and loaded script does not have to be saved in the nonvolatile memory of the Series 3700 before it can be run. However, an unsaved script will be lost when the Series 3700 is turned off.

Saving a named script

Only a named script can be saved in nonvolatile memory of the Series 3700. After creating and loading a named script, use one of the following commands to save it.

```
myscript.save()           -- To save script in internal
    memory
myscript.save('filename') -- To save script on flash drive
```

Where:

- `myscript` is the user-defined name of the script.
- `filename` is a filename for the script to save it on a user-supplied USB flash drive.

The `save` command will save the script to internal nonvolatile memory if no filename is specified. If a script is not saved in nonvolatile memory, the script will be lost when the Series 3700 is turned off.

The `myscript.save()` command saves the script under the original name that was created and loaded. The `myscript.save('filename')` shell command is used to save the script to a user-supplied USB flash drive. If you save the script to a filename that already exists, it will be overwritten. The filename may have an absolute or relative path to the current working directory. If using an absolute path, include `/usb1/` at the beginning to denote the flash drive. Also, if the filename includes an extension, it must be `.tsp`; otherwise, an error occurs. If no extension is specified, the `.tsp` extension will be added.

Examples:

- Assume a script named "test1" has been created and loaded. The following command saves the script in nonvolatile memory:

```
test1.save()
```

- To save the script named "test1" under a filename ("test2") on a user-supplied USB flash drive, send the following command:

```
test1.save('/usb1/test2.tsp')
```

- To save the anonymous script, provide a valid name for the name attribute of the anonymous script. Once the anonymous script is named, it can be saved. For example, to save the anonymous script as "MyTest":

```
script.anonymous.name = 'MyTest'
script.anonymous.save()
```

- To execute anonymous script that was just saved as 'MyTest':

```
script.user.scripts.MyTest()
```

- The anonymous script may be saved to the flash drive without setting the name attribute. For example, to save the anonymous script on the flash drive as "MyAnonTest.tsp":

```
script.anonymous.save('/usb1/MyAnonTest.tsp')
```

Loading this file ('MyAnonTest.tsp') back into the unit from the flash drive will cause an existing anonymous script to be overwritten.

Loading a user script

The following command is used to create a script from a specified file:

```
myscript = script.load(file, name)
```

Where:

- **myscript** is the created script, or `nil` if an error occurred. If the name parameter is an empty string, or if the name is absent or `nil`, and the script name cannot be extracted from the file, this will be the only handle to the created script.
- **file** is the absolute or relative path and filename of the script file to import.
- **name** (optional) is the name of the script to be created. The script's name attribute is initialized to this value, which (if not an empty string) also serves as the key used to access the script through the `script.user.scripts` table.

If the `name` parameter is present (not `nil`), any script name embedded in the file is ignored. Also, if `name` conflicts with the name of an existing script in the `script.users.scripts` table, the existing script will be unnamed (that is, its name attribute will be set to the empty string) before it is replaced in the `script.user.scripts` table by the newly-created script.

If the name parameter is absent or `nil`, the command attempts to extract the script's name from the file. Any conflict between the extracted name and that of an existing script in the `script.user.scripts` table generates an error. If the script name cannot be extracted, the created script's name attribute is initialized to the empty string, and must be set to a valid non-empty string before saving the script to internal memory.

The file to be uploaded must contain the `loadscript` or `loadandrunscript` keywords, the script's body, and the `endscript` keyword.

Examples:

To load a file called "MyUserList.tsp" from the USB flash drive and name the script "myuserlist," send the following command:

```
mylist = script.load('MyUserList.tsp', 'myuserlist')
```

To execute the myuserlist script:

```
script.user.scripts.myuserlist()  
mylist()
```

Running a user script

Running an unnamed script

There can only be one unnamed (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. On the front panel, an unnamed script appears as <anonymous>. Use one of the following commands to execute the chunk of the last loaded anonymous script. The following four commands perform the same operation.

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

Running a named script

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

```
myscript()  
myscript.run()  
script.user.scripts.myscript()  
script.user.scripts.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 commands execute the chunk of the script.

```
test3()  
script.user.scripts.test3()
```

Running scripts automatically

Scripts can be set to run automatically when the Series 3700 is turned on. You can assign one or more [Autorun scripts](#) (on page 2-26) and one [Autoexec script](#) (on page 2-26).

Autorun scripts

When a saved script is set to autorun, it will automatically load and run when the Series 3700 is turned on. Any number of scripts can be set to 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 one of the following autorun attributes to "yes." Setting it to "no" disables autorun.

```
myscript.autorun
script.user.scripts.myscript.autorun
```

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

Make sure to save the script in nonvolatile memory after setting the `autorun` attribute.

Example:

Assume a script named "test5" is in the run-time environment. Set the script to autorun as follows:

```
test5.autorun = "yes" or script.user.scripts.test5.autorun
= "yes"
test5.save()
```

The next time the Series 3700 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 it, set the `autorun` attribute to "no" and save the script.

Autoexec script

One script can be designated as the `autoexec` script. When the Series 3700 is turned on, the `autoexec` script will start after all the autorun scripts have run.

```
loadscript autoexec
loadandrunscript autoexec
```

Form an `autoexec` script 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 nonvolatile memory. See [Creating a user script](#) (on page 2-20) for details on creating a script.

Running a user script from the Series 3700 front panel

Use the following commands to enter or delete a name in the User menu option from the LOAD key:

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

Where:

- `displayname` is the name to be added to (or deleted from) the User menu.
- `chunk` is the name of the chunk (Lua executable code).

It does not matter what order the items are added to the User menu, as they 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 script name to the User menu for the chunk as follows:

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

To run the chunk from the front panel:

1. Press the **LOAD** key.
2. Select **USER** and press the **ENTER** key.
3. Select the user chunk from list and press the **ENTER** key. The chunk is loaded for front panel execution.

NOTE If you're used to using `print` in Test Script Builder, note that the output of the prints using this procedure will not function the same as when you're in Test Script Builder. You may find that it makes more sense to use Test Script Builder to get the output you need.

4. Press the **RUN** key to execute.

To run a script directly without adding it to the USER menu:

1. Press the **LOAD** key.
2. Select **SCRIPTS** and press the **ENTER** key. There may be a short pause before a menu is displayed that represents the scripts in the instrument.
3. Select the script from the list and press the **ENTER** key. Now the script is loaded for front panel execution.

NOTE If you're used to using `print` in Test Script Builder, note that the output of the prints using this procedure will not function the same as when you're in Test Script Builder. You may find that it makes more sense to use Test Script Builder to get the output you need.

4. Press the **RUN** key to execute.

Loading a script from the Series 3700 front panel

To load a script from a USB flash drive:

1. Press the **MENU** key to open the main menu.
2. Select the **SCRIPT** option.
3. Select the **LOAD** option. A menu is displayed that includes the USB option.
4. Select the **USB** option. A menu is displayed that lists the .tsp files and directories on the flash drive. If you select a directory, a new menu is displayed that lists the .tsp files and directories in that directory.
5. Selecting a .tsp file will cause the system to attempt to load the file.
 - If the file is not a valid script file, an error message is posted and no further action is taken.
 - Loading an anonymous script will overwrite the existing anonymous script.
 - A file that does not contain `loadscript` and `endscript` shell keywords will be loaded as an anonymous script.
 - The display will indicate if a named script already exists in memory. If it does, you will be prompted to overwrite the script. The display returns to step 4 if an error occurs or if you select "No" when prompted to overwrite the script. Otherwise, the display proceeds to step 6 after indicating that the script loaded successfully.
 - The script is loaded using the name that follows the `loadscript` shell keyword instead of filename. Also, the script is loaded into the `script.user.scripts` table.
6. The SCRIPT ACTION menu lists the options of `ACTIVE_FOR_RUN` or `SAVE_INTERNAL`. Proceed to step 7 if **ACTIVE_FOR_RUN** is selected or step 8 if **SAVE_INTERNAL** is selected.
7. **ACTIVE_FOR_RUN** associates the script with the **RUN** button if you selected **YES**. The script replaces the active executable chunk selected under the **LOAD** button. If **NO** is selected, the display returns to step 6.
8. If you select **SAVE_INTERNAL**, you will be prompted to save the script into internal memory. The ICL equivalent is `myscript.save()` with no parameters. If you select **YES**, proceed to step 5 of [Saving a script from the Series 3700 front panel](#) (on page 2-29). If **NO** is selected, the display returns to step 6. Note that anonymous scripts cannot be saved internally.

Saving a script from the Series 3700 front panel

To save a script to internal memory or a USB flash drive:

1. Press the **MENU** key to bring up the Main Menu.
2. Select the **SCRIPT** option.
3. Select the **SAVE** option to bring up a menu listing the scripts available to save from the `script.user.scripts` table. This may take several seconds before displaying.
4. Select the script that you want to save. It may be the anonymous script or one of the user-named scripts.
5. Select where you want to save the script, either in internal memory or on the USB flash drive. You cannot save the anonymous script to internal memory. Only named scripts can be saved internally.
 - Select **INTERNAL** to save the script to internal memory using the script's name attribute.
 - Select **USB**, and a prompt will display for a filename using the first 13 characters of the name attribute as a default name followed by a modifiable 3-digit number. If the filename exists, the word (overwrite) appears on the display. The file is automatically saved with a `.tsp` extension.

Modifying a user script

You can modify a user script stored in nonvolatile memory by retrieving the script listing, which can then be modified, loaded, and saved in nonvolatile memory. See [Retrieving a user script](#) (on page 2-30) for details.

NOTE If you are using the Test Script Builder to modify a user script stored in nonvolatile memory, retrieve the script listing from the Project Navigator.

Script management

Retrieving a user script

You can retrieve the source code contained in a user script from nonvolatile memory, which can then be modified and saved as a user script under the same name or a new name.

NOTE You can load a modified user script into the Series 3700 using the same name or a new name.

The following command returns a catalog listing of the user scripts stored in the Series 3700:

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

Example:

To retrieve the catalog listing for user scripts:

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

The following function retrieves the source code of a saved script. 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:

To retrieve the source of a saved script named "test7":

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

NOTE To see the contents of the anonymous script, use the `script.list.anonymous.list()` command.

Deleting a script from the system

To completely remove a script from the system, all references to the script will need to be deleted from the run-time environment. A script may be removed from the run-time environment by assigning it to `nil`. For example, to remove the script named "test8" from the run-time environment, send the following code to set `test8` to a `nil` value:

```
test8 = nil
```

To delete a script from the `script.user.scripts` table, set the `name` attribute to an empty string(""), which makes the script nameless. This does not make the script become the anonymous script. For example, to remove the script named "test8" from the `script.user.scripts` table, send the following code to set the `name` attribute for `test8` as an empty string:

```
script.user.scripts.test8.name = ""
```

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

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

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

Example:

To delete a user script named "test8" from nonvolatile memory:

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

Restoring a script in the run-time environment

A script is inherently a global variable that 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 nonvolatile 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:

To restore a user script named "test9" from nonvolatile memory:

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

Differences: Remote versus local state

The Series 3700 can be in either the local state or the remote state.

- When it is in the local state (REM annunciator off), the instrument is operated using the front panel controls.
- When it is in the remote state (REM annunciator on), instrument operation is controlled by the computer.

When the instrument is powered-on, it is in the local state.

Remote state

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

- Sending a command from the computer to the instrument
- Running a script (a 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
- Web-control (logging in to the Series 3700 web interface)

While the instrument is in the remote state, front panel controls are disabled. However, the LOCAL key will be active unless it has not been explicitly locked out by the user program. When an interactive script is running, use the activated front panel controls 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 (a USER test) is completed, the instrument will return to the local state.

TSP-Link™ system

A test system can be expanded to include up to 64 instruments that are enabled using TSP-Link. The system can be stand-alone or PC-based.

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 primary node and can control all of the other nodes, which become its subordinates. When the script is finished running, all the nodes in the system return to local operation (REM annunciators turn off), and the primary/subordinate relationship between nodes is dissolved.

PC-based system

When using a computer, the GPIB, LAN, or USB interface to any single node becomes the interface to the entire system. When a command is sent through one of these interfaces, all nodes go into remote operation (REM annunciators turn on).

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

Test Script Language (TSL) Reference

Introduction

A script is a program written using the Test Script Language (TSL) that the Test Script Processor (TSP) executes. TSL is an efficient language, with simple syntax and extensible semantics. TSL is derived from the Lua programming language. See the website for the [Lua Programming Language](http://www.lua.org) (<http://www.lua.org>) for more information. Another source of useful information is [lua-users](http://lua-users.org) (<http://lua-users.org>), created for and by users of Lua programming language.

Lua programming language 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.

<code>var = nil</code>	<code>var</code> is <code>nil</code> .
<code>var = 1.0</code>	<code>var</code> is now a number.
<code>var = 0.3E-12</code>	<code>var</code> is still a number.
<code>var = 7</code>	<code>var</code> is still a number.
<code>var = "Hello world!"</code>	<code>var</code> is now a string.
<code>var = "I said, Hello world!"</code>	<code>var</code> is still a string.
<code>var = function(a, b) return(a+b) end</code>	<code>var</code> is now a function that adds two numbers.
<code>var = {1, 2., 3.00e0}</code>	<code>var</code> is now a table (in other words, an 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 non-value that represents the absence of a useful value.

Operators

Arithmetic operators:

- `+` (addition)
- `-` (subtraction)
- `*` (multiplication)
- `/` (division)
- `-` (negation)

Relational operators:

<	(less than)
>	(greater than)
<=	(less than or equal)
>=	(greater than or equal)
~=	(not equal)
==	(equal)

Logical operators:

and
or
not

Functions

TSL allows you to define functions which can take a pre-defined number of parameters and return multiple parameters. Functions are first-class values in TSL, which means that they can be stored in variables, passed as arguments, and returned as results if desired.

This is a function called "add_two":

```
function add_two(parameter1, parameter2)
  return(parameter1 + parameter2)
end
print(add_two(3, 4))
```

This is an alternate syntax for defining a function, called "add_three":

```
add_three = function(parameter1, parameter2, parameter3)
  return(parameter1 + parameter2 + parameter3)
end
print(add_three(3, 4, 5))
```

This function 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)
```

The function's output is:

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

Tables/arrays

TSL makes extensive use of the data type "table," which is a very flexible array-like data type.

To define a table:

```
-- A table with four elements, which are numbers.
atable = {1, 2, 3, 4}
```

Send the following commands to print it:

```
-- Tables are indexed on one, NOT zero. atable[index] is
  true if there is an element at that index. nil is
  returned otherwise. 0 does NOT evaluate to false, only
  nil does.
i = 1

Index into table using a number.
while atable[i] do
  print (atable[i])
  i = i + 1
end
```

The command's output is:

```
1
2
3
4
```

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

Below is a table used to create a circle pseudoclass. 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}

-- Index using a string; print the clr property.
print(circle['clr'])

-- Index using a string; print the diam property.
print(circle['diam'])

-- Change the diam element by calling setdiam.
circle['setdiam'](2)

-- Print the diameter of the circle; circle['diam'] is a
  simpler syntax of circle.diam
print(circle.diam)

-- Change the diameter of the circle again.
circle.setdiam(3)

-- Print diam property again using simple syntax.
print(circle.diam)

```

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 -----
--
-- This is if expression 1. Zero is true in Lua language;
   this is a contrast to C language, where 0 evaluates
   false. In TSL, nil is false and everything else is true.
if 0 then
  print("Zero is true!")
else
  print("Zero is false.")
end
x = 1
y = 2

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

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

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

This is if expression 5.
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
```

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 or until an expression evaluates to false.

The following code contains an example iteration:

```
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 -----  
--  
-- Will exit when list[element] = nil  
print("Counting elements in list")  
print("on numeric index")  
element = 1  
while list[element] do  
    list[element] = nil  
    print(element, list[element])  
    element = element + 1  
end
```

```
--
----- REPEAT loop -----
--
print("Counting 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, and more. TSL also includes instrument control extension libraries, which provide programming interfaces to the instrumentation accessible by the TSP™. These libraries are automatically loaded when the TSP starts and do not need to be managed by the programmer.

Base library functions

<code>print(x)</code>	Prints the argument <code>x</code> to the active host interface, using the <code>tostring()</code> function to convert <code>x</code> to a string.
<code>collectgarbage([limit])</code>	Sets the garbage-collection threshold to the given limit (in Kbytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, then TSL immediately runs the garbage collector. If the limit parameter is absent, it defaults to 0 (thus forcing a garbage-collection cycle). See NOTE below for more information.
<code>gcinfo()</code>	Returns the number of Kbytes of dynamic memory that TSP™ is using.
<code>tonumber(x [,base])</code>	Returns <code>x</code> converted to a number. If <code>x</code> is already a number, or a convertible string, then the number is returned; otherwise, it returns <code>nil</code> . 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 <code>A</code> (in either upper or lower case) represents 10, <code>B</code> represents 11, and so forth, with <code>Z</code> 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.
<code>tostring(x)</code>	Receives an argument of any type and converts it to a string in a reasonable format.
<code>type(v)</code>	Returns the type of its only argument, coded as a string. The possible results of this function are: <code>nil</code> , <code>number</code> , <code>Boolean</code> , <code>table</code> , or <code>function</code> .
<p>NOTE TSL does automatic memory management, which means you do not have to allocate memory for new objects and free it when the objects are no longer needed. TSL manages memory automatically by occasionally running a garbage collector 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.</p> <p>TSL uses two numbers to control its garbage-collection cycles. One number counts how many bytes of dynamic memory TSL is using; the other is a threshold. When the number of bytes crosses the threshold, TSL runs the garbage collector, which reclaims the memory of all "dead" objects. The byte counter is adjusted, and then the threshold is reset to twice the new value of the byte counter.</p>	

String library functions

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

```
-- Returns the internal numerical code of the i-th
   character of string s, or nil if the index is out of
   range.
string.byte(s [,i])

-- 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.char(i1, i1, ...)

-- 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, l, 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.format(fs, e1, e2, ...)

-- Returns the length of the strings.
string.len(s)

-- Returns a copy of the string s with all uppercase
   letters changed to lowercase.
string.lower(s)

-- Returns a string that is the concatenation of n copies
   of the string s.
string.rep(s, n)

-- 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.sub(s, i [,j])

-- Returns a copy of the string s with all lowercase
   letters changed to uppercase.
string.upper(s)
```

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.

Function	Definition
<code>math.abs(x)</code>	Returns the absolute value of the argument <code>x</code> .
<code>math.acos(x)</code>	Returns the principal value of the trigonometric arc cosine function of <code>x</code> .
<code>math.asin(x)</code>	Returns the principal value of the trigonometric arc sine function of <code>x</code> .
<code>math.atan(x)</code>	Returns the principal value of the trigonometric arc tangent function of <code>x</code> .
<code>math.atan2(y, x)</code>	Returns the principal value of the trigonometric arc tangent function of <code>y/x</code> .
<code>math.ceil(x)</code>	Returns the smallest floating-point number not less than <code>x</code> whose value is an exact mathematical integer.
<code>math.cos(x)</code>	Returns the trigonometric cosine function of <code>x</code> .
<code>math.deg(x)</code>	Returns the value of <code>x</code> in degrees, where <code>x</code> is in radians.
<code>math.exp(x)</code>	Returns the exponential function of <code>x</code> ; that is, e^x , where e is the base of the natural logarithms.
<code>math.floor(x)</code>	Returns the largest floating-point number not greater than <code>x</code> whose value is an exact mathematical integer.
<code>math.log(x)</code>	Returns the natural logarithm function of <code>x</code> .
<code>math.log10(x)</code>	Returns the base-10 logarithm function of <code>x</code> .
<code>math.max(x, y, ...)</code>	Returns the maximum value of its numeric argument(s).
<code>math.min(x, y, ...)</code>	Returns the minimum value of its argument(s).
<code>math.mod(x, y)</code>	Returns an approximation to the mathematical value f such that f has the same sign as <code>x</code> , the absolute value of f is less than the absolute value of <code>y</code> , and there exists an integer k such that $k*y+f = x$.
<code>math.pi</code>	Variable containing the value of π (3.141592654).
<code>math.pow(x, y)</code>	Returns x^y .
<code>math.rad(x)</code>	Returns the value of <code>x</code> in radians, where <code>x</code> is in degrees.
<code>math.sin(x)</code>	Returns the trigonometric sine function of <code>x</code> .
<code>math.sqrt(x)</code>	Returns the non-negative square root of <code>x</code> .
<code>math.tan(x)</code>	Returns the trigonometric tangent function of <code>x</code> .

Function	Definition
<code>math.frexp()</code>	Splits x into a fraction f and exponent n , such that f is 0.0 or $0.5 \leq f \leq 1.0$, and $f * 2^n$ is equal to x . Both f and n are returned: $f, n = \text{math.frexp}(x)$.
<code>math.ldexp(x, n)</code>	Returns the inverse of the <code>math.frexp()</code> function; it computes the value $x * 2^n$.
<code>math.random([x], [y])</code>	When called without an argument, returns a pseudo-random real number in the range $[0, 1]$. When called with number x , returns a pseudo-random integer in the range $[1, n]$. When called with two arguments, x and y , returns a pseudo-random integer in the range $[x, y]$.
<code>math.randomseed(x)</code>	Sets a "seed" for the pseudo-random generator. Equal seeds produce equal sequences of numbers.

TSP Advanced Features

In this section:

Introduction	3-1
Using groups to manage nodes on TSP-Link™ network.	3-4
Running parallel test scripts	3-6
Using the data queue for real-time communication	3-8
Copying test scripts across the TSP-Link™ network	3-8
Removing stale values from the reading buffer	3-9
Commands related to TSP advanced features	3-10

Introduction

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.

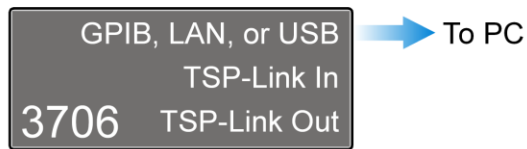
There are two methods you can use to run test scripts in parallel:

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

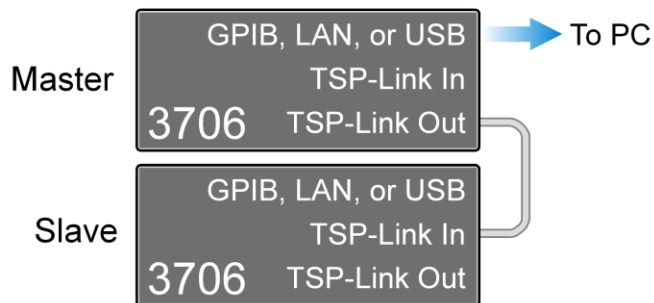
The following figure displays the first method, which consists of multiple TSP-Link networks. Each TSP-Link network has a master node and a GPIB connection to the PC.

Figure 3-1: Multiple TSP-Link networks

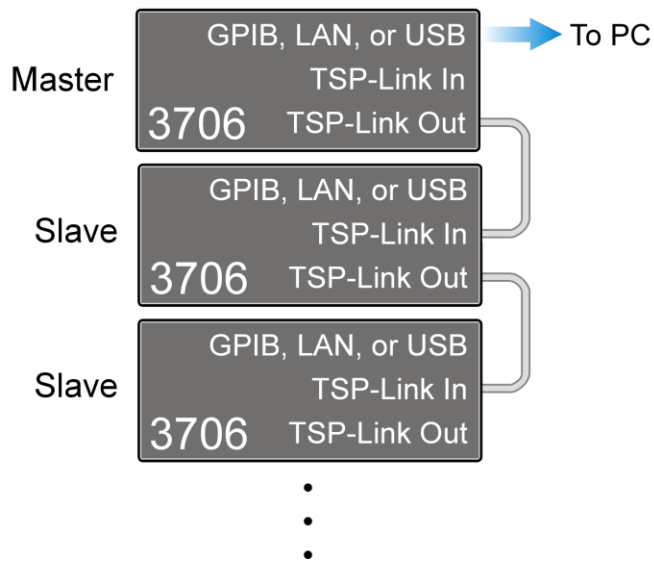
Single-instrument TSP-Link network



Two-instrument TSP-Link network



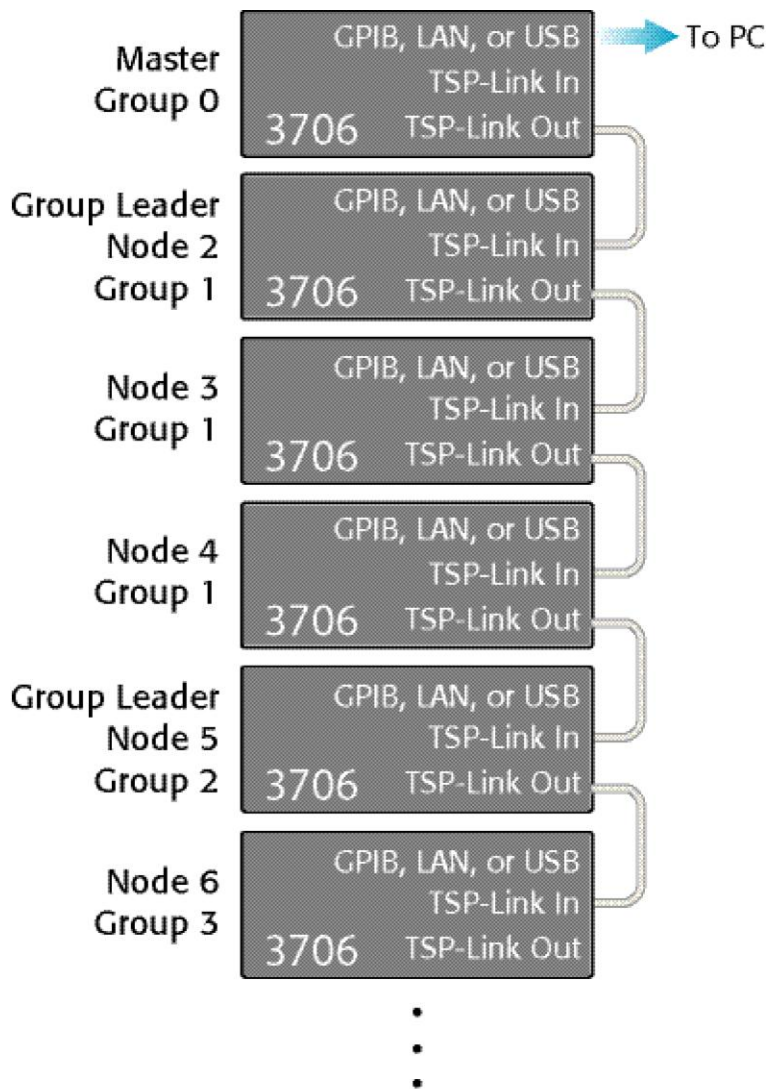
TSP-Link network with more than three instruments



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).

The following figure displays a TSP-Link network with groups. This method requires one TSP-Link network and a single GPIB connection to the PC.

Figure 3-2: Single TSP-Link network with groups



The following table 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 (see the previous figure).

TSP-Link network group functions

Group number	Group members	Current function
0	Master node	<ul style="list-style-type: none"> 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
1	Group leader Node 2	<ul style="list-style-type: none"> Runs the test script initiated by the master node Initiates remote operations on Node 3
	Node 3	<ul style="list-style-type: none"> Performs remote operations initiated by Node 2
2	Group leader Node 5	<ul style="list-style-type: none"> Runs the test script initiated by the master node Initiates remote operations on Node 4
	Node 4	<ul style="list-style-type: none"> Performs remote operations initiated by Node 5
3	Group leader Node 6	<ul style="list-style-type: none"> Runs the test script initiated by the master node

TSP-Link™ has three synchronization lines that function similar to the Digio synchronization lines. See [digio functions and attributes](#) (on page 13-11) and [Hardware trigger modes](#) (on page 8-18) for more detailed information.

Using groups to manage nodes on TSP-Link™ network

The primary purpose of 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 members. Group numbers are not assigned automatically; you must use the [Instrument Control Library \(ICL\)](#) (on page 12-1) 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 send the `execute` command on a remote node
- Cannot initiate remote operations on any node in a remote group if any node in that remote group is performing an overlapped operation
- Sends the `waitcomplete` command to wait for the local group that the master node belongs to, to wait for a remote group, or to wait for all nodes on the TSP-Link network to complete overlapped operations

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 send the `waitcomplete` command without a parameter to wait for all nodes assigned to the same group number

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:

- Each time the node powers off, the group number for that node changes to 0
- 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

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

Reassigning groups

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

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

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 3-8) 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 on the TSP-Link™ network
- **Master node only:** To wait for all groups to complete overlapped operations

For additional information, see [waitcomplete\(\)](#) (on page 3-16).

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

```
-- Waits for each node in group N to complete all
   overlapped operations.
waitcomplete(N)
-- 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 all
   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 3700 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 view the capacity, see the [Instrument Control Library \(ICL\)](#) (on page 12-1).

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 with the same name:

Note the following:

- 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 return stale values from the reading buffer to the cached data. 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 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 = dmm.makebuffer(20) " ..
               "dmm.measure.count = 20 " ..
               "dmm.measure(rbremote)")

-- 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("dmm.measure(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 will never update. Every time the print
  command is issued after the first print command, the
  same data buffer values will print.
print(rblocal[1])
```

Commands related to TSP advanced features

dataqueue.add()	
Function	Store an item of data in the data queue.
Usage	<pre>success = dataqueue.add(value[, timeout])</pre> <p>value: The data item to add.</p> <p>timeout: Maximum amount of time in seconds to wait for room in the queue if it is full.</p> <p>success: Success indication.</p>
Remarks	<p>This function will add one entry to the data queue. If the queue is full, this function will wait up to <code>timeout</code> seconds for room to be made available. This function will return true if the value was added to the data queue. It will return false if the queue is full and the item could not be added before the timeout expires.</p> <p>The timeout value may only be specified when called from the local node. If a timeout value is not given, the function will not wait for room in the queue if it is full.</p> <hr/> <p>NOTE If <code>value</code> is a table, this function will make a deep copy of it rather than storing a reference to it. This will make a complete copy of all entries within the table, including all nested tables.</p>

dataqueue.CAPACITY	
Attribute	The maximum number of entries the data queue can hold.
Usage	<pre>capacity = dataqueue.CAPACITY</pre> <p>capacity: Maximum number of entries in the data queue.</p>
Remarks	This constant indicates the maximum number of values that can be stored in the data queue.

dataqueue.clear()	
Function	Clear the data queue.
Usage	<pre>dataqueue.clear()</pre>
Remarks	This function will remove all entries from the data queue. If any nodes are waiting to add data to the queue, this method will force them to fail as if they timed out.

dataqueue.count	
Attribute	The number of entries currently stored in the data queue.
Usage	<pre>count = dataqueue.count</pre> <p>count: Number of entries in the data queue.</p>

dataqueue.count	
Remarks	This attribute is a read-only attribute that indicates how many entries are in the data queue.

dataqueue.next()	
Function	Retrieve an entry from the data queue.
Usage	<pre>value = dataqueue.next([timeout])</pre> <p>timeout: Maximum amount of time in seconds to wait for data if the queue is empty. value: The next entry from the data queue.</p>
Remarks	<p>This function will remove the next entry from the data queue and return its value. If the queue is empty, this function will wait up to <code>timeout</code> seconds for data to arrive. If no data arrives before the timeout expires, this function will return nil.</p> <p>The timeout value may only be specified when called from the local node. If a timeout value is not given, the function will not wait for data to be put in the queue if it is empty.</p> <p>NOTE If the entry is a table, this function will return a deep copy of its contents at the time the table was added to the data queue rather than returning a reference to the original table.</p>

localnode.execute()	
Function	Execute TSL code.
Usage	<pre>localnode.execute(chunk)</pre> <p>chunk: Source TSL code to execute.</p>
Remarks	<p>This function will execute the given TSL code.</p> <p>NOTE This command cannot actually be used on the local node. It is provided for the sole purpose of executing scripts on this node from a remote master node. The <code>localnode</code> prefix to the command is an artifact of command organization and how remote commands are shared between nodes.</p>

localnode.getglobal()	
Function	Get a the value of a global variable.
Usage	<pre>value = localnode.getglobal(name)</pre> <p>name: The global variable name. value: The value of the variable.</p>

localnode.getglobal()	
Remarks	<p>This function will return the value of the given global variable. This function is provided to allow code running on a remote master node to retrieve values of variables from that node. This function should not be used to retrieve the value of a global variable on the local node when using the local node as the master. Accessing the variable directly is far more efficient.</p> <hr/> <p>NOTE This command is provided for the sole purpose of accessing variables on this node from a remote master node. The <code>localnode</code> prefix to the command is an artifact of command organization and how remote commands are shared between nodes.</p>
localnode.setglobal()	
Function	Set a the value of a global variable.
Usage	<pre>localnode.setglobal (name, value)</pre> <p>name: The global variable name to create. value: The value to assign to the variable.</p>
Remarks	<p>This function will assign the given value to a global variable. This function is provided to assign values to variables from a remote master node. This function should not be used to assign values to global variables on the local node when using the local node as the master, assigning the value directly is far more efficient.</p> <hr/> <p>NOTE This command is provided for the sole purpose of accessing variables on this NOTE from a remote master node. The <code>localnode</code> prefix to the command is an artifact of command organization and how remote commands are shared between nodes.</p>
tsplink.group	
Attribute	The group number of a TSP-Link™ node.
Usage	<pre>groupnumber = tsplink.group</pre> <pre>tsplink.group = groupnumber</pre> <p>groupnumber: The TSP-Link group number for the node.</p>
Remarks	This attribute controls the TSP-Link group number used for DTNS. Set this attribute to zero to remove the node from all subgroups.
tsplink.master	
Attribute	The node number of the master node.
Usage	<pre>master = tsplink.master</pre> <p>master: The node number of the master node.</p>
Remarks	This read-only attribute indicates which node is the master node.

tsplink.trigger[N].assert()	
Function	Simulates the occurrence of the trigger and generates the corresponding event id.
Usage	<code>tsplink.trigger[N].assert()</code> N : The trigger line to assert (1–3).
Remarks	This function will generate a trigger pulse on the given TSP-Link™ trigger line.
Also see	tsplink.trigger[N].clear() (on page 3-13) tsplink.trigger[N].mode (on page 3-13) tsplink.trigger[N].overrun (on page 3-15) tsplink.trigger[N].release() (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)
tsplink.trigger[N].clear()	
Function	Clear the event detector for a trigger.
Usage	<code>tsplink.trigger[N].clear()</code> N : The trigger line (1–3).
Remarks	A trigger's event detector remembers if an event has been detected since the last <code>tsplink.trigger[N].wait</code> call. This function clears a trigger's event detector and discards the previous history of the trigger line.
Also see	tsplink.trigger[N].mode (on page 3-13) tsplink.trigger[N].overrun (on page 3-15) tsplink.trigger[N].release() (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)
tsplink.trigger[N].mode	
Attribute	The trigger operation/detection mode.
Usage	To read trigger operation/detection mode: <code>mode = tsplink.trigger[N].mode</code> To write trigger operation/detection mode: <code>tsplink.trigger[N].mode = mode</code> N : The trigger line (1–3). mode : Trigger mode.

tsplink.trigger[N].mode	
Remarks	<p>This attribute controls the mode in which the trigger event detector as well as the output trigger generator will operate on the given trigger line. mode can be one of the following values:</p> <p>tsplink.TRIG_BYPASS Allow direct control of the line.</p> <p>tsplink.TRIG_EITHER Detect rising or falling edge triggers as input. Assert a TTL-low pulse for output.</p> <p>tsplink.TRIG_FALLING Detect falling edge triggers as input. Assert a TTL-low pulse for output.</p> <p>tsplink.TRIG_RISING Use digio.TRIG_RISINGA if the line is in the high output state. Use digio.TRIG_RISINGM if the line is in the low output state.</p> <p>tsplink.TRIG_RISINGA Detect rising edge triggers as input. Assert a TTL-low pulse for output.</p> <p>tsplink.TRIG_RISINGM Assert a TTL-high pulse for output. Input edge detection is not possible in this mode.</p> <p>tsplink.TRIG_SYNCHRONOUS Detect falling edge triggers as input and latch them low. Assert a TTL-low pulse for output.</p> <p>tsplink.TRIG_SYNCHRONOUSA Detect falling edge triggers as input and automatically latch and drive them low when detected. Release a latched line for output.</p> <p>tsplink.TRIG_SYNCHRONOUM Detect rising edge triggers as input. Assert a TTL-low pulse for output.</p>
Remarks, continued	<p>The default trigger mode for a line will be TRIG_BYPASS. In this mode, the line can be directly controlled as a digital I/O line. When programmed to any other mode, the output state of the I/O line is controlled by the trigger logic and the user-specified output state of the line will be ignored.</p> <p>For compatibility with older firmware, when the trigger mode is set to TRIG_RISING, the user specified output state of the line will be examined. If the output state selected when the mode is changed is high, the actual mode used will be TRIG_RISINGA. If the output state selected when the mode is changed is low, the actual mode used will be TRIG_RISINGM.</p>

tsplink.trigger[N].mode	
Also see	tsplink.trigger[N].assert (on page 3-12) tsplink.trigger[N].clear() (on page 3-13) tsplink.trigger[N].overrun (on page 3-15) tsplink.trigger[N].release() (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)

tsplink.trigger[N].overrun	
Attribute	Event detector overrun status.
Usage	<pre>overrun = tsplink.trigger[N].overrun</pre> <p>overrun: Trigger overrun state. N: The trigger line (1–3).</p>
Remarks	<p>This attribute is a read-only attribute. It indicates if an event was ignored due to the event detector being in the detected state when the event occurred. This is an indication of the state of the event detector built into the synchronization line itself.</p> <p>This attribute does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the event.</p>
Also see	tsplink.trigger[N].assert (on page 3-12) tsplink.trigger[N].clear() (on page 3-13) tsplink.trigger[N].mode (on page 3-13) tsplink.trigger[N].release() (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)

tsplink.trigger[N].release()	
Function	Release a latched trigger.
Usage	<pre>tsplink.trigger[N].release()</pre> <p>N: The trigger line (1–3).</p>
Remarks	This function will release a latched trigger on the given TSP-Link™ trigger line.
Also see	tsplink.trigger[N].assert (on page 3-12) tsplink.trigger[N].clear() (on page 3-13) tsplink.trigger[N].mode (on page 3-13) tsplink.trigger[N].overrun (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)

tsplink.trigger[N].wait()	
Function	Wait for a trigger.
Usage	<pre>triggered = tsplink.trigger[N].wait(timeout)</pre> <p>N: The trigger line (1–3). timeout: Maximum amount of time in seconds to wait for the trigger. triggered: Trigger detection indication.</p>
Remarks	<p>This function will wait for an input trigger. If one or more trigger events were detected since the last time <code>tsplink.trigger[N].wait()</code> or <code>tsplink.trigger[N].clear()</code> (on page 3-13) was called, this function will return immediately.</p> <p>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.</p>

waitcomplete()	
Function	Wait for all overlapped commands to complete.
Usage	<pre>waitcomplete([group])</pre> <p>group: Optional TSP-Link™ group on which to wait.</p>
Remarks	<p>This function will wait for all overlapped operations within given group to complete. A group number may only be specified from the master node. If no group is specified, the local group will be used. If zero is given for the group, this function will wait for all nodes in the system.</p> <hr/> <p>NOTE Any nodes that are not assigned to a group (their group number is zero) will be considered to be part of the master's group.</p> <hr/> <p>This function will wait for all previously started overlapped commands to complete. Currently the Series 3700 has no overlapped commands implemented. However, other TSP™-enabled products like the Series 3700 has overlapped commands. Therefore, when the Series 3700 is a TSP master to a slave device with overlapped commands, use this function to wait until all overlapped operations are completed.</p>

Using the Front Panel

In this section:

Front panel introduction	4-1
Display	4-4
Special keys and power switch	4-11
Operation keys	4-17
Range keys, cursor keys, and navigation wheel	4-32
Action keys.....	4-33

Front panel introduction

This section describes the Keithley Instruments Series 3700 System Switch/Multimeter front panels.

The menu options under the CHAN key and CONFIG CHAN menus vary, depending on the channel type of the selected channel. When selecting a range of channels, all channel types might need to match for some operations like read and write of a Digital I/O channel or opening and closing of Switch channels. Some operations, like `reset()`, work with a range of mix channel types.

NOTE Not all models will have a digital multimeter (DMM) installed. All DMM-related documentation is not applicable to those models.

If your model does not have a front panel, see the following sections to make the appropriate changes.

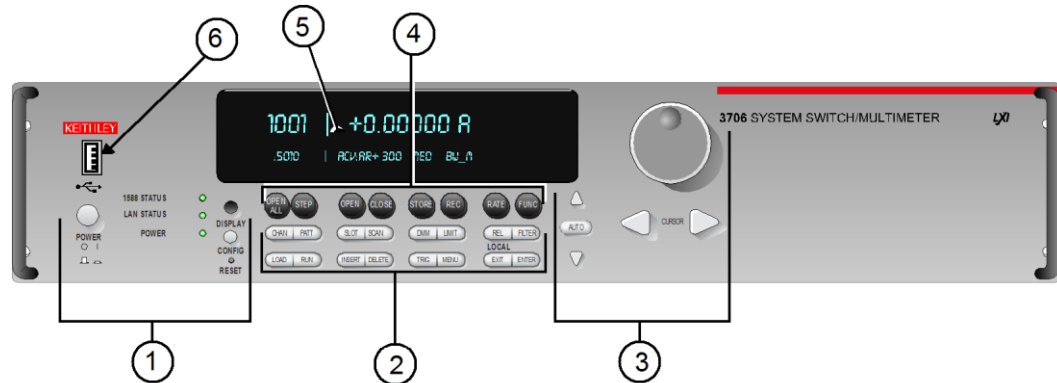
- GPIB address with `gpib.address` (on page 13-187) command.
- LAN configuration using [LAN functions and attributes](#) (on page 13-190).

Use the menu system to scan channels, with the following limitations:

- You can add digital I/O and totalizer channels to a front panel scan list for reading by using the INSERT key (ICL equivalent of `scan.add()` (on page 13-230)).
- You cannot add digital I/O and totalizer channels to a front panel scan list for writing (ICL equivalent of `scan.addwrite()` (on page 13-231)).
- You cannot add DAC channels to a front panel scan list for writing or reading.
- Pressing the INSERT key with a DAC channel selected generates an error message.

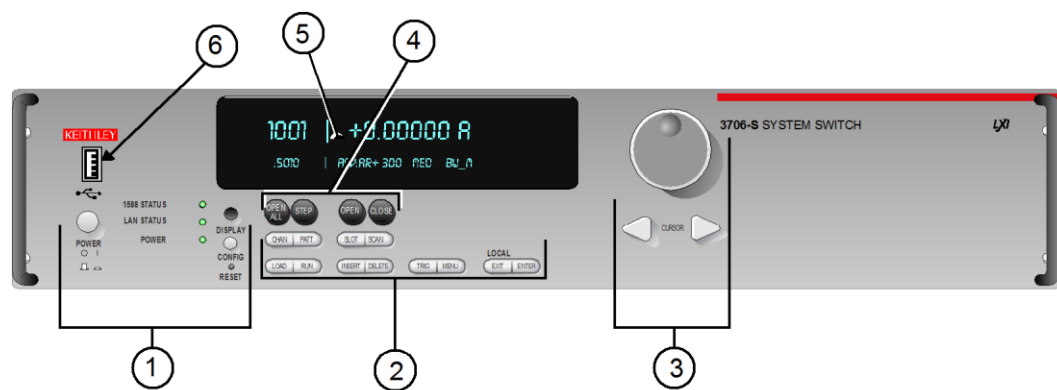
NOTE Only widths of one are supported by the front panel when reading or writing to a Digital I/O channel.

Figure 4-1: Model 3706 System Switch/Multimeter



Item	Description
1	Special keys and power switch (on page 4-11)
2	Operation keys (on page 4-17)
3	Range keys, cursor keys, and navigation wheel (on page 4-32)
4	Action keys (on page 4-33)
5	Display (on page 4-4)
6	USB connector

Figure 4-2: Model 3706-S System Switch (no DMM)



NOTE To see current settings for LAN, see the applicable `lan.status.*` commands (for example, to see the present IP address of the Series 3700, send the following command: `lan.status.ipaddress` (on page 13-201)).

Figure 4-3: Model 3706-NFP System Switch/Multimeter

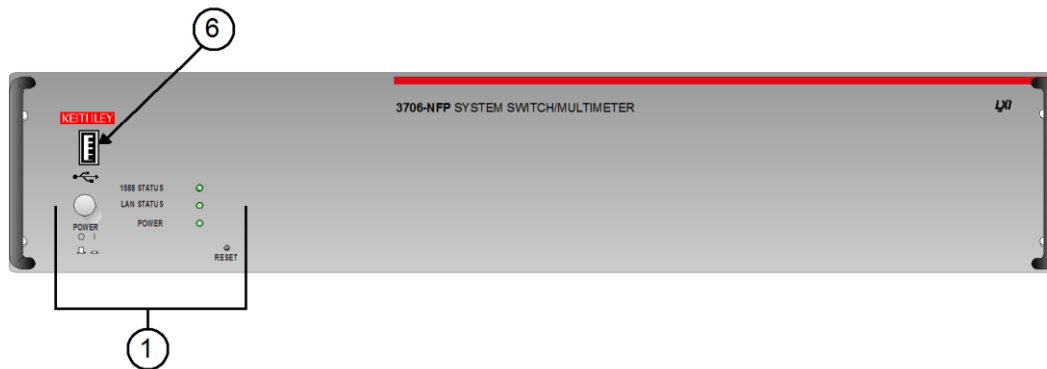
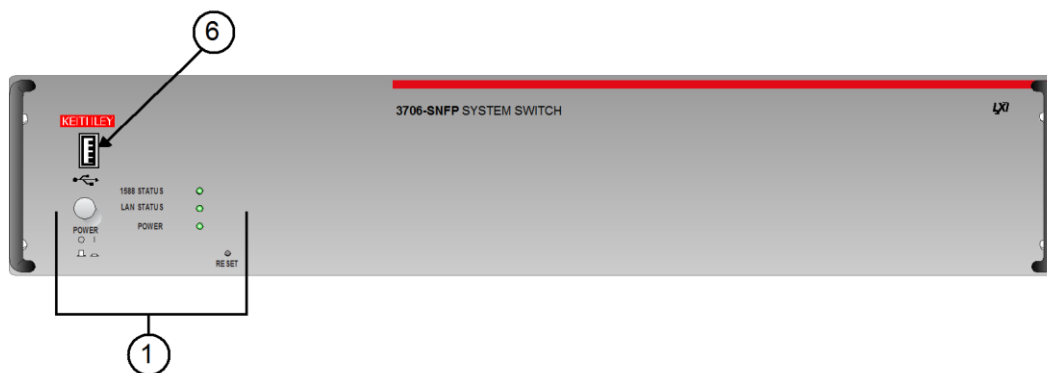


Figure 4-4: Model 3706-SNFP System Switch (no DMM)



Display

The Series 3700 display provides visual information on the present active channel, including the channel selected (or range or pattern), channel state, last DMM measurement reading (where applicable), DMM settings (where applicable), and error indications. The display and the navigation wheel provide a way to change the active channel or channel ranges, as well as access, view, and edit the various menus and menu items.

The Series 3700 has three LEDs on the front panel; these LEDs represent 1588 status, LAN status, and power.

- When you turn on the unit, the power LED is illuminated.
- When the instrument is connected through the Ethernet with no errors, the LAN status LED is illuminated. However, the LAN status LED is off when the instrument is not connected through the Ethernet or there is a connection problem.
- When you press the identify option (ID button) on the home web page for the Series 3700, the LAN status LED blinks.
- The 1588 status LED indicates 1588 operation. When this LED is off, the 1588 feature is disabled or improperly configured.
- The LED blinks at a one-second rate when the instrument is the 1588 master. If the instrument is a slave, the LED will not blink.

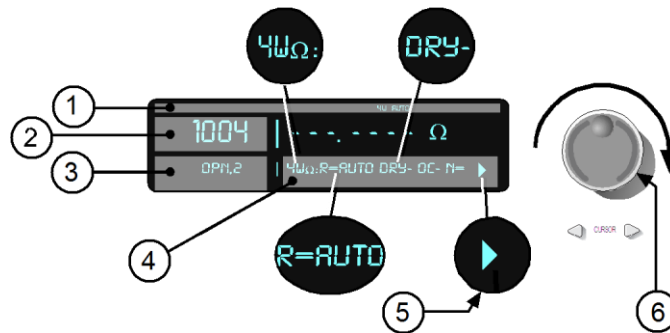
See the following figure (labeled "Active channel display example") for an active channel example. The 4WΩ and AUTO range annunciators are lit (1). Also, the active channel is 1004 (Slot 1, Channel 004) (2). The present state of the channel is open and it has two poles (3). The present state of the attributes for this channel (4) are:

- 4WΩ function set for AUTO range
- Dry-circuit ohms disabled (DRY-)
- Offset compensation off (OC-).

Other attributes, such as NPLC, are available for this specific active channel (1004) as indicated by arrow (5) being lit. View them by turning the navigation wheel (6) to scroll through the attribute list.

NOTE Access attribute and menu lists that are larger than the display by turning the navigation wheel (6). Displayed arrows (5) indicate additional attributes or menu items (as applicable) that are available by turning the navigation wheel (6) in the direction the arrow points. If an arrow (5) is not displayed, there are no additional menu choices in that direction. Switch-only systems have none of these features.

Figure 4-5: Active channel display example



The top line of the display (1) contains the following annunciators:

Annunciator	Description
* (asterisk)	Readings are being stored in the selected reading buffer. This is OFF when no buffer is selected or the selected buffer is full.
4W	Displays 4-wire resistance or RTD temperature reading.
AUTO	Auto range enabled for the selected DMM function.
EDIT	Unit in edit mode (for front panel).
FILT	Filter enabled for the selected DMM function.
LSTN	Instrument addressed to listen over GPIB.
MATH	mX+b, percent, or reciprocal (1/X) calculation enabled for the selected DMM function.

Annunciator	Description
REL	Relative enabled for selected DMM function.
REM	Instrument in bus remote mode or web interface control mode (all interfaces, LAN, GPIB, or USB).
SRQ	Service request over GPIB.
TALK	Instrument addressed to talk over GPIB bus.
TRIG	ON when the front panel has requested a reading from the DMM. OFF when the reading is finished.

The bottom left line of the display (4) contains the DMM attribute symbols. The symbols that appear are dependent on whether the attribute exists for the selected function. The following table indicates the DMM attribute symbols that may appear on the front panel. If the symbol has a value associated with it, the third column in the table indicates the value definition.

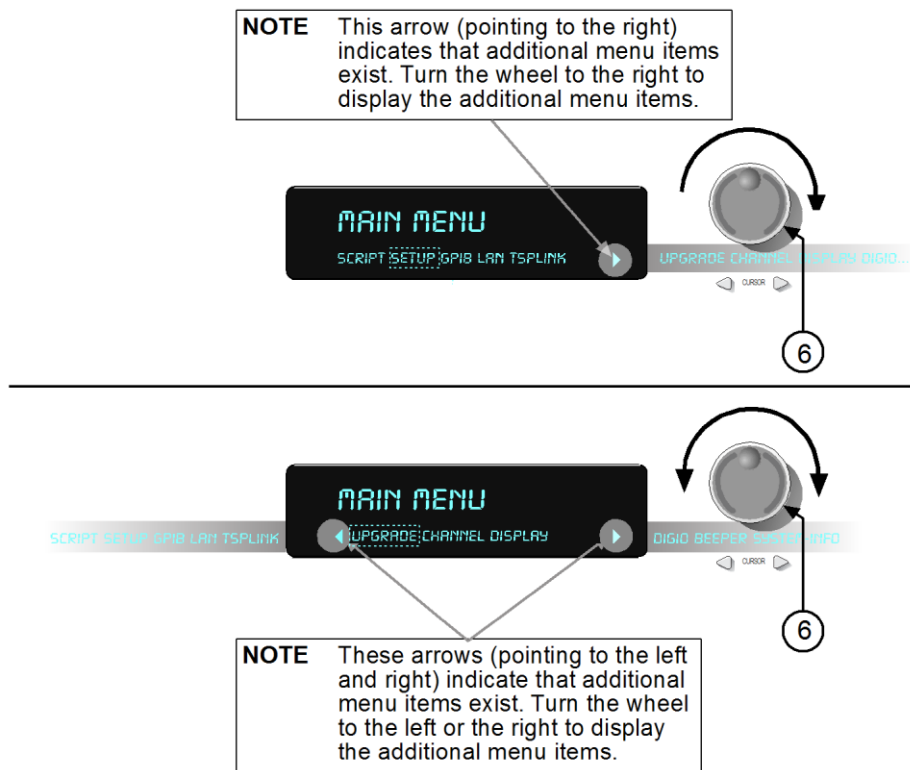
Front panel DMM attribute	Symbol	Values
range	R=	AUTO or n, where n equals the range
nplc	N=	n, where n equals the NPLC
auto delay	AD	+ for ON, 1 for ONCE, or 0 for OFF
auto zero	AZ	+ for ON or – for OFF
line sync	LS	+ for ON or – for OFF
limit	LIM	+ for a limit enabled or – for limits disabled
detector bandwidth	DBW	3, 30, or 300
threshold	THR=	n, where n indicates the threshold
aperture	A=	n, where n indicates the aperture setting
dry circuit	DRY	+ for ON or – for OFF
offset compensation	OC	+ for ON or – for OFF
thermocouple sensor K	K_T/C	N/A
thermocouple sensor T	T_T/C	N/A
thermocouple sensor E	E_T/C	N/A
thermocouple sensor R	R_T/C	N/A
thermocouple sensor S	S_T/C	N/A
thermocouple sensor B	B_T/C	N/A
thermocouple sensor N	N_T/C	N/A
thermistor	THRM	N/A
three-wire RTD	3RTD	N/A
4-wire RTD	4RTD	N/A
simulated reference junction	RJ_SIM	N/A

Front panel DMM attribute	Symbol	Values
internal reference junction	RJ_INT	N/A
external reference junction	RJ_EXT	N/A

NOTE To access the main menu, press the **MENU** key.

See the following figure for a menu example. In the example, the MAIN MENU is displayed. Turn the navigation wheel (6) or press the **CURSOR** keys to scroll through the available menu items. In the following figure's first display, there is a right arrow indicator. This indicates there are additional menu items to the right. In the figure's second display, both right and left arrows are active, indicating that there are additional items in both directions. To select the highlighted (flashing) menu item, press the navigation wheel or press the **ENTER** key.

Figure 4-6: MAIN MENU display



Channel type indication

When selecting channels, the following information is displayed, which indicates the channel type on the front panel.

- For switch channels, the name of the assigned DMM configuration (for example, “nofunction”, “dcvolts”, “my4wire”, and so on) is displayed and below it, the channel state (for example, OPN or CLS, along with 2 or 4 for pole setting) is displayed.
- For digital I/O channels, DIGITAL IO appears and below the channel number is DIG IN or DIG OUT to indicate the channel mode.
- For totalizer channels, TOTALIZER appears and below the channel number is Fall Ed or Rise Ed to indicate mode edge.
- For DAC channels, DAC OUTPUT appears and below the channel number is pv1, off, and so on to indicate protection, function, and output enable.

Using the front panel with non-switch channels

To read a value from the main front panel screen, select the channel and press TRIG. To see a digital I/O channel in hex format (instead of normal binary) use CONFIG+TRIG.

A star symbol (*) or exclamation point symbol (!) may appear after the reading. The meaning of the symbol depends on channel type.

- A star symbol (*) appears after the reading to indicate that the reading matches the MATCH setting for digital I/O and totalizer channels.
- An exclamation point symbol (!) appears after the reading to indicate an overload state condition on that channel for digital I/O and DAC channels.
- An exclamation point symbol (!) appears after the reading to indicate an overflow state condition on a totalizer channel.
- If the power state is OFF for totalizer or DAC channels, the display shows “DISABLED” instead of any readings.

The following table lists the front panel channel attributes that indicate the various channel mode settings (ICL equivalent `channel.setmode()` (on page 13-77)), channel output enable (ICL equivalent `channel.setoutputenable()` (on page 13-78)), and channel label (ICL equivalent `channel.setlabel()` (on page 13-74)). Some of the attributes have alternate symbols, depending on the operation performing on front panel and whether it is being used with the 6 or 12 character label symbol.

- For digital I/O and totalizer channels, the label symbol is listed first, followed by a comma and then mode symbols. If the label is the factory default setting, then only the mode is listed.
- For DAC channels, the label symbol is listed first, followed by a comma and then mode symbols, followed by another comma and the output enable symbol. If the label is the factory default setting, then only the mode and output enable symbols are listed.

Front Panel Channel Setting	Symbol	Definition	Symbol meaning
Channel label	XXXXXX	First 6 characters of label	Used with single letter symbols
	XXXXXXXXXX XX	First 12 characters of label	Used with the non-single letter symbols
Digital I/O mode settings	DIG IN	Digital input mode	Used with 12-character label or no label
	DIG OUT	Digital output mode	Used with 12-character label or no label
	DIG pOUT	Digital output protected mode	Used with 12 character label or no label

Front Panel Channel Setting	Symbol	Definition	Symbol meaning
	I (uppercase "i")	Digital input mode	Used with 6-character label
	O	Digital output mode	Used with 6-character label
	P	Digital output protected mode	Used with 6-character label
Totalizer mode settings	Rise Ed	Totalizer rising edge mode	Used with 12-character label or no label
	Fall Ed	Totalizer falling edge mode	Used with 12-character label or no label
	Rise-TTL	Totalizer rising edge TTL level mode	Used with 12-character label or no label
	Fall-TTL	Totalizer falling edge TTL level mode	Used with 12-character label or no label
	Rise-RST	Totalizer rising edge read reset mode	Used with 12-character label or no label
	Fall-RST	Totalizer falling edge read reset mode	Used with 12-character label or no label
	RiseTRST	Totalizer rising edge TTL read reset mode	Used with 12-character label or no label
	FallTRST	Totalizer falling edge TTL read reset mode	Used with 12-character label or no label
	R	Totalizer rising edge mode	Used with 6-character label
	F	Totalizer falling edge mode	Used with 6-character label
DAC mode settings	V	Voltage function mode	Used with 6-character label
	I (uppercase "i")	Current function either 1 or 2 mode	Used with 6-character label
	V1	Voltage function 1 mode	Used with 12-character label or no label
	I1	Current function 1 mode	Used with 12-character label or no label
	I2	Current function 2 mode	Used with 12-character label or no label
	pV1	Protected voltage function 1 mode	Used with 12-character label or no label
	pI1	Protected current function 1 mode	Used with 12-character label or no label

Front Panel Channel Setting	Symbol	Definition	Symbol meaning
	pl2	Protected current function 2 mode	Used with 12-character label or no label
DAC output enable settings	Off	Output enable is disabled	Used with 6 or 12 character label
	On	Output enable is enabled	Used with 6 or 12 character label

Special keys and power switch

CONFIG key

Press the **CONFIG** key to access an attribute menu that enables you to configure channels, channel patterns, DMM functions or settings, reading buffers, scans, and other operations. Refer to the following for more information:

- [CHAN key configuration](#) (see "CONFIG CHAN key - SWITCH channel type" on page 4-11)
- [PATT key configuration](#) (on page 4-30)
- [SCAN key configuration](#) (on page 4-31)
- [DMM key configuration](#) (on page 4-21)
- [LIMIT key configuration](#) (on page 4-27)
- [REL key configuration](#) (on page 4-30)
- [FILTER key configuration](#) (on page 4-25)

CONFIG CHAN key

CONFIG CHAN key - SWITCH channel type

Press the **CONFIG** key and then the **CHAN** key to open the CHANNEL ATTRibute menu. If you press the **CHAN** key when a pattern is selected, the unit goes into channel selection mode.

When changing attribute settings for a range of channels, the menu option for the first channel specified in the range is highlighted. For example, selecting Channels 3 to 5 on Slot 3 on the front panel (3003:3005) as a range shows the current attribute setting for 3003 when an attribute menu is displayed. When the attribute setting is selected for a range, the entire range of channels is updated to that value. To view or set an individual attribute setting for only one channel, be sure to select a single channel range. For example, 3003:3003 would only affect Channel 3 on Slot 3, which is displayed as 3003 with the channel state and poles setting below it.

The CHAN ATTR menu contains:

- **LABEL:** Sets the label associated with the specified channel. From the front panel, the label can be up to 12 characters. Remotely, the label may be up to 20 characters. This option will not be displayed if multiple channels are selected. Related ICL command: [channel.setlabel\(\)](#) (on page 13-74).
- **BACKPLANE:** Opens the BACKPLANE menu. Use this menu to add or remove backplane channels from the specified channels. Related ICL command: [channel.setbackplane\(\)](#) (on page 13-70).
- **FORBID:** Allows or prevents the closing of the specified channels. Related ICL commands: [channel.setforbidden\(\)](#) (on page 13-73) and [channel.clearforbidden\(\)](#) (on page 13-36).
- **POLE:** Sets the number of poles for the specified channels. Related ICL command: [channel.setpole\(\)](#) (on page 13-79).
- **DELAY:** Sets additional delay time for the specified channels. Related ICL command: [channel.setdelay\(\)](#) (on page 13-72).
- **COUNT:** Displays closure cycles for the specified channel. This option is not displayed if multiple channels are selected. Related ICL command: [channel.getcount\(\)](#) (on page 13-48).
- **DMM_CONFIG:** Sets the DMM configuration associated with the specified channels. Related ICL command: [dmm.setconfig\(\)](#) (on page 13-168).

CONFIG CHAN key - DIGIO channel type

Press the **CONFIG CHAN** key to open the DIGIO ATTR menu. The DIGIO ATTR menu is not available when a range of channels is selected. If a range is selected, pressing CONFIG CHAN displays the following:

- DIGIO ATTR MENU
- <No Edit by Range, Use EXIT>

Therefore, to see the following options, select a single DIGIO channel.

LABEL

Enter up to 12 characters for the label for a channel. Related ICL command: [channel.setlabel\(\)](#) (on page 13-74).

DELAY

Enter the value for the delay in 1ms steps from 0 to 60 seconds for a channel. Related ICL command: [channel.setdelay\(\)](#) (on page 13-72).

MODE

Sets the mode attribute on a channel. Select one of the following options:

- INPUT
- OUTPUT
- OUTPUT_PROTECTED

Related ICL command: [*channel.setmode\(\)*](#) (on page 13-77).

MATCH

Sets the match value on a channel. Enter the value as 8-bit binary. Related ICL command: [*channel.setmatch\(\)*](#) (on page 13-76).

MATCH_TYPE

Sets the match type on a channel. Select one of the following options:

- EXACT
- ANY
- NOT_EXACT
- NONE

Related ICL command: [*channel.setmatchtype\(\)*](#) (on page 13-76).

STATE

Queries for the state of a channel and displays the value in the top line, labeled by `STATE=`. Related ICL command: [*channel.getstate\(\)*](#) (on page 13-56).

CONFIG CHAN key - TOTALIZER channel type

Press the **CONFIG CHAN** key to open the TOTAL ATTR menu. The TOTAL ATTR menu is not available when a range of channels is selected. If a range is selected, pressing CONFIG CHAN displays the following:

- TOTAL ATTR MENU
- <No Edit by Range, Use EXIT>

Therefore, to see the following options, select a single Totalizer channel.

LABEL

Enter up to 12 characters for the label for a channel. Related ICL command: [*channel.setlabel\(\)*](#) (on page 13-74).

MODE

Sets the mode attribute on a channel. Select one of the following options:

- EDGE. Indicates the edge for the Totalizer channel to increment its count. Select from one of the following options:
 - FALLING
 - RISING
- THRESHOLD. Indicates the threshold range. Select from one of the following options:
 - TTL
 - NON_TTL
- RESET. Indicates if the count value gets reset after being read. Select from one of the following options:
 - ON
 - OFF

Related ICL command: [*channel.setmode\(\)*](#) (on page 13-77).

MATCH

Sets the match value on a channel. Enter the value between 0 and 65535.

Related ICL command: [*channel.setmatch\(\)*](#) (on page 13-76).

MATCH_TYPE

Sets the match type on a channel. Select one of the following options:

- EXACT
- ANY
- NOT_EXACT
- NONE

Related ICL command: [*channel.setmatchtype\(\)*](#) (on page 13-76).

STATE

Queries for the state of a channel and displays the value in the top line, labeled by `STATE=`. Related ICL command: [*channel.getstate\(\)*](#) (on page 13-56).

POWER

Sets the power state attribute on a channel. Select one of the following options:

- ENABLE
- DISABLE

Related ICL command: [channel.setpowerstate\(\)](#) (on page 13-80)

CONFIG CHAN key - DAC channel type

Press the **CONFIG CHAN** key to open the DAC ATTR menu. The DAC ATTR menu is not available when a range of channels is selected. If a range is selected, pressing CONFIG CHAN displays the following:

- DAC ATTR MENU
- <No Edit by Range, Use EXIT>

Therefore, to see the following options, select a single DAC channel.

NOTE If the DAC channel has power set to DISABLE, then the menu choices change to only show the option to change the power setting, until the power is set to ENABLE.

LABEL

Enter up to 12 characters for the label for a channel. Related ICL command: [channel.setlabel\(\)](#) (on page 13-74).

DELAY

Enter the value for the delay in 1ms steps from 0 to 60 seconds for a channel. Related ICL command: [channel.setdelay\(\)](#) (on page 13-72).

MODE

Sets the mode attribute on a channel. Select one of the following options:

- FUNCTION. Sets the desired function for a channel. Select one of the following options:
 - VOLTAGE
 - CURRENT_1
 - CURRENT_2
- PROTECT. Indicates if the protection mode for a channel is enabled. Select one of the following options:
 - AUTO
 - OFF

Related ICL command: [channel.setmode\(\)](#) (on page 13-77).

OUTPUT

Sets the output enable attribute on a channel. Select one of the following options:

- ENABLE
- DISABLE

Related ICL command: [channel.setoutputenable\(\)](#) (on page 13-78)

STATE

Queries for the state of a channel and displays the value in the top line, labeled by STATE=. Related ICL command: [channel.getstate\(\)](#) (on page 13-56).

POWER

Sets the power state attribute on a channel. Select one of the following options:

- ENABLE
- DISABLE

Related ICL command: [channel.setpowerstate\(\)](#) (on page 13-80)

DISPLAY key

Press this key to toggle between the main and user display modes.

POWER switch

Press this switch to turn the Series 3700 on (I). Press it again to turn the Series 3700 off (O).

RESET switch

Use the **RESET** switch to restore the Series 3700 factory default LAN settings. Refer to the [LAN functions and attributes](#) (on page 13-190) (`lan.config.x`, where `x` represents the specific command) for factory default information.

Operation keys

CHAN key

Different menus are displayed for switch channel types or non-switch channel types when the CHAN key is pressed.

CHAN key - switch channel type

Press the **CHAN** key to open the CHANNEL ACTION menu.

The CHANNEL ACTION menu contains the following items:

- **OPEN:** This menu item opens the specified channels for switching aspects. Related Instrument Control Library (ICL) command: [channel.open\(\)](#) (on page 13-59).
- **CLOSE:** This menu item closes specified channels. These closures are appended to the already closed channels. Related ICL command: [channel.close\(\)](#) (on page 13-37).
- **EXCLOSE:** This menu item closes the specified channels so that they are exclusively closed. Related ICL command: [channel.exclusiveclose\(\)](#) (on page 13-41).
- **EXSLOTCLOSE:** This menu item exclusively closes specified channels on the specified slots. Related ICL command: [channel.exclusiveslotclose\(\)](#) (on page 13-43).
- **RESET:** This menu item resets specified channels to factory default settings. Resetting a channel deletes any channel patterns that contain that channel. Related ICL command: [channel.reset\(\)](#) (on page 13-68).

CHAN key - DIGIO channel type

Press the **CHAN** key to open the DIGIO ACTION menu. Unless noted, the menu option supports a range of selected channels.

READ

Displays a value from a channel as 8-bit binary. This menu option does not appear if a range of channels is selected. Related ICL command: [channel.read\(\)](#) (on page 13-67).

NOTE Only widths of one are supported by the front panel when reading or writing to a Digital I/O channel.

WRITE

Writes a value to a channel. Enter the value as 8-bit binary. Related ICL command: [channel.write\(\)](#) (on page 13-83).

NOTE Only widths of one are supported by the front panel when reading or writing to a Digital I/O channel.

RESET_STATE

Resets the channel state. Related ICL command: [channel.resetstatelatch\(\)](#) (on page 13-70).

RESET

Restores the factory default settings of selected channels or all channels. Related ICL command: [channel.reset\(\)](#) (on page 13-68).

CHAN key - TOTALIZER channel type

Press the **CHAN** key to open the TOTAL ACTION menu. Unless noted, the menu option supports a range of selected channels.

READ

Displays a value from a channel as a number between 0 and 65535. This menu option does not appear if a range of channels is selected. Related ICL command: [channel.read\(\)](#) (on page 13-67).

WRITE

Writes a value to a channel. Enter the value between 0 and 65535. Related ICL command: [channel.write\(\)](#) (on page 13-83).

RESET_STATE

Resets the channel state. Related ICL command: [channel.resetstatelatch\(\)](#) (on page 13-70).

RESET

Restores the factory default settings of selected channels or all channels.

Related ICL command: [channel.reset\(\)](#) (on page 13-68).

CHAN key - DAC channel type

Press the **CHAN** key to open the DAC ACTION menu. Unless noted, the menu option supports a range of selected channels.

NOTE If the DAC channel has power set to DISABLE, then the menu choices change to only show the option to change the power setting, until the power is set to ENABLE.

READ

Displays a value from a channel. This menu option does not appear if a range of channels is selected.

A number is displayed that is dependent on the channel's selected mode function, as well as the card model of the selected channel. For example, a number from one of the following ranges is displayed for a DAC channel on the 3750, based on the channel's selected mode function:

- -12 to +12 for MODE-FUNCTION as VOLTAGE_1
- 0 to 20 mA for MODE-FUNCTION as CURRENT_1
- 4 to 20 mA for MODE-FUNCTION as CURRENT_2

Related ICL command: [channel.read\(\)](#) (on page 13-67).

WRITE

Writes a value from a channel. This menu option does not appear if a range of channels is selected.

A number is displayed that is dependent on the channel's selected mode function, as well as the card model of the selected channel. For example, a number from one of the following ranges is displayed for a DAC channel on the 3750, based on the channel's selected mode function:

- -12 to +12 for MODE-FUNCTION as VOLTAGE_1
- 0 to 20 mA in 1 uA steps for MODE-FUNCTION as CURRENT_1
- 4 to 20 mA in 1 uA steps for MODE-FUNCTION as CURRENT_2

Related ICL command: [channel.write\(\)](#) (on page 13-83)

RESET_STATE

Resets the channel state. Related ICL command: [channel.resetstatelatch\(\)](#) (on page 13-70).

RESET

Restores the factory default settings of selected channels or all channels. Related ICL command: [channel.reset\(\)](#) (on page 13-68).

DELETE key

Press the **DELETE** key to delete the first occurrence of the selected channel(s) or channel pattern (including function) from the scan list. If a selected item is not contained in the scan list, no error is reported.

To remove all occurrences of a channel from the list, keep pressing the **DELETE** key.

To view the current scan list after deleting items:

1. Press the **SCAN** key when on the main display.
2. Select the **LIST** option and press the **ENTER** key.
3. Use the navigation wheel or **CURSOR** keys to scroll through the list.

DMM key

Press the DMM key to open the DMM ACTION menu.

The DMM ACTION menu contains the following items:

- **MEASURE:** Takes measurements on the digital multimeter (DMM) without using the trigger model. Related ICL command: [dmm.measure\(\)](#) (on page 13-150).
- **COUNT:** Indicates the number of measurements to take when a measurement is requested. Related ICL command: [dmm.measurecount](#) (on page 13-151).
- **LOAD:** Recalls a user or factory DMM configuration. Use the navigation wheel to scroll through available configurations. Related ICL command: [dmm.configure.recall\(\)](#) (on page 13-128).
- **SAVE:** Creates a DMM configuration with the pertinent attributes based on the selected function, and associates it with the specified name. Related ICL command: [dmm.configure.set\(\)](#) (on page 13-129).
- **OPEN:** Opens the specified channel and/or channel pattern. Related ICL command: [dmm.open\(\)](#) (on page 13-154).
- **CLOSE:** Closes the specified channel or channel pattern in preparation for a DMM measurement. Related ICL command: [dmm.close\(\)](#) (on page 13-123).
- **RESETFUNC:** Returns the DMM aspects of the system for only the active function to factory default settings. Related ICL command: [dmm.reset\(\)](#) (on page 13-161).
- **RESETALL:** Returns all DMM functions of the instrument to the factory default settings. Related ICL command: [dmm.reset\(\)](#) (on page 13-161).

DMM key configuration

Press the **CONFIG** key and then the **DMM** key to open a DMM attribute menu for the active function. For example, if the DCV function is active, pressing the **CONFIG** key and then the **DMM** key opens the DC VOLT ATTR menu.

Each function only has access to the applicable attributes for that function. Brief definitions of the available attributes are contained in the following paragraphs. Refer to the appropriate ICL for additional attribute information in [Instrument Control Library \(ICL\)](#) (on page 12-1).

APERTURE

Configures the aperture setting for the active DMM function in seconds. Related ICL command: [dmm.aperture](#) (on page 13-110).

AUTODELAY

Configures the auto delay setting for the active DMM function. Related ICL command: [dmm.autodelay](#) (on page 13-112).

AUTORANGE

Configures the auto range setting for the DMM. Related ICL command: [dmm.autorange](#) (on page 13-113).

AUTOZERO

Configures the auto zero setting for the DMM. Related ICL command: [dmm.autozero](#) (on page 13-114).

DBREF

Configures the DB reference setting for the DMM in volts. Related ICL command: [dmm.dbreference](#) (on page 13-131).

DETECTBW

Configures the detector bandwidth setting for the selected DMM function. For more information, see [Bandwidth](#) (on page 5-7). Related ICL command: [dmm.detectorbandwidth](#) (on page 13-132).

DIGITS

Configures the display digits setting for the selected DMM function. For more information, see [Digits ICL programming](#) (on page 5-4). Related ICL command: [dmm.displaydigits](#) (on page 13-132).

DRYCIRCUIT

Configures the dry circuit setting for the selected DMM function. Related ICL command: [dmm.drycircuit](#) (on page 13-133).

FILTER

Opens the FILTER menu for the selected DMM function. See [FILTER key configuration](#) (on page 4-25).

FUNC

Displays a menu that allows you to scroll through the available DMM functions. Use the navigation wheel or **CURSOR** keys to scroll the menu options and press **ENTER** as soon as the desired function is highlighted. Related ICL command: [dmm.func](#) (on page 13-137).

INPUTDIV

Enables or disables the 10M Ω input divider. Related ICL command: [dmm.inputdivider](#) (on page 13-140).

LIMIT

Opens the LIMIT menu for the selected DMM function. See [LIMIT key configuration](#) (on page 4-27).

LINESYNC

Enables or disables line sync during measurements.

Related ICL command: [dmm.linesync](#) (on page 13-144).

MATH

Selecting the **MATH** menu item opens the MATH MENU. Items contained in this menu are:

- **ENABLE:** Enables or disables math operation on measurements. Related ICL command: [dmm.math.enable](#) (on page 13-147).
- **FORMAT:** Specifies the math operation to perform on measurements. Related ICL command: [dmm.math.format](#) (on page 13-147).
- **BFACTOR:** Specifies the offset for the $y = mX + b$ operation. Related ICL command: [dmm.math.mxb.bfactor](#) (on page 13-148).
- **MFACTOR:** Specifies the scale factor for the $y = mX + b$ operation. Related ICL command: [dmm.math.mxb.mfactor](#) (on page 13-149).
- **MXBUNITS:** Specifies the unit character for the $y = mX + b$ operation. Related ICL command: [dmm.math.mxb.units](#) (on page 13-149).
- **PERCENT:** Specifies the constant to use for the percent operation. Related ICL command: [dmm.math.percent](#) (on page 13-150).

For more information, see:

- [mX+b](#) (on page 6-4)
- [Reciprocal \(1/X\)](#) (on page 6-7)
- [Percent](#) (on page 6-6)

NPLC

Configures the integration rate in line cycles for the DMM. Related ICL command: [dmm.nplc](#) (on page 13-152).

OFFSETCOMP

Configures the offset compensation setting for the DMM. Related ICL command: [dmm.offsetcompensation](#) (on page 13-153).

OPENDETECT

Configures the state of the thermocouple or 4-wire ohms open detector that is being used. Related ICL command: [dmm.opendetector](#) (on page 13-155).

RANGE

Configures the range of DMM for the selected function for one channel type. For more information, see [Range](#) (on page 5-1). Related ICL command: [dmm.range](#) (on page 13-156).

REL

Opens the REL menu for the selected DMM function. See [REL key configuration](#) (on page 4-30).

THERMO

Selecting the **THERMO** menu item opens the THERMO menu. Items contained in this menu are:

- **REFJUNCT:** Allows selection of the reference junction to use. Available choices are: SIMULATED, EXTERNAL, or INTERNAL. Related ICL command: [dmm.refjunction](#) (on page 13-157).
- **SIMREF:** Specifies the simulated reference temperature for thermocouples. Related ICL command: [dmm.simreftemperature](#) (on page 13-169).
- **THERMISTOR:** Specifies the type of thermistor. Related ICL command: [dmm.thermistor](#) (on page 13-170).
- **THERMOCOUPLE:** Specifies the thermocouple type. Related ICL command: [dmm.thermocouple](#) (on page 13-171).
- **TRANSDUCER:** Selects the transducer type (THERMOCOUPLE, THERMISTOR, 3RTD, or 4RTD). Related ICL command: [dmm.transducer](#) (on page 13-173).
- **THREERTD:** Specifies the type of 3-wire RTD. Related ICL command: [dmm.threertd](#) (on page 13-172).
- **FOURRTD:** Specifies the type of 4-wire RTD. Related ICL command: [dmm.fourrtd](#) (on page 13-137).
- **USER:** Specifies USER type of RTD (ALPHA, BETA, DELTA, or ZERO). Related ICL commands: [dmm.rtdalpha](#) (on page 13-162), [dmm.rtdbeta](#) (on page 13-163), [dmm.rtddelta](#) (on page 13-164), [dmm.rtdzero](#) (on page 13-165).

THRESHOLD

Configures the threshold range. Related ICL command: [dmm.threshold](#) (on page 13-173).

UNITS

Configures the units for voltage and temperature measurements. Related ICL command: [dmm.units](#) (on page 13-174).

ENTER key

Press the **ENTER** key to accept the current selection or bring up the next menu options.

NOTE Pressing the navigation wheel performs the same function as the **ENTER** key.

EXIT key

Press the **EXIT** key to:

- Cancel the selection and to return to the previous menu display.
- Exit remote operation.
- Abort a scan that is running.
- Abort a script that is executing.

FILTER key

Press the **FILTER** key to enable and disable the filter for selected function. When the filter is enabled, the FILT annunciator will light. See [Filter](#) (on page 5-8) for more information.

FILTER key configuration

Press the **CONFIG** key and then the **FILTER** key to open the FILTER menu.

The FILTER menu contains the following menu items:

- **ENABLE:** Enables or disables filtered measurements for the selected DMM function. Related ICL command: [dmm.filter.enable](#) (on page 13-134).
- **COUNT:** Indicates the filter count setting for the selected DMM function. Related ICL command: [dmm.filter.count](#) (on page 13-134).
- **TYPE:** Indicates the filter averaging type for the DMM measurements on the selected DMM functions (MOVING or REPEAT). Related ICL command: [dmm.filter.type](#) (on page 13-135).
- **WINDOW:** Indicates the filter window for the DMM measurements (0 to 10% in 0.1% increments). Related ICL command: [dmm.filter.window](#) (on page 13-136).

FUNCTION key

Each press of the **FUNC** key immediately configures the DMM for the next function in the list:

- **dcvolts**: DC voltage
- **acvolts**: AC voltage
- **dccurrent**: DC current
- **accurrent**: AC current
- **twowireohms**: 2-wire ohm (resistance)
- **fourwireohms**: 4-wire ohm (resistance)
- **commonsideohms**: Common-side ohm (resistance)
- **frequency**: Frequency
- **period**: Period
- **continuity**: Continuity
- **temperature**: Temperature

For example, if the DMM function is configured for dcvolts, pressing the **FUNC** key four times will configure the DMM for acvolts, then for dcurrent, then for accurrent, and then finally for twowireohms, which ends up as the active function on the DMM. If you do not want the DMM to be momentarily configured for the other functions while getting to desired one then, press the **CONFIG** key followed by the **FUNC** key. Next, scroll to the desired function and press the **ENTER** key when the desired function is highlighted (blinking). Related ICL command: [dmm.func](#) (on page 13-137).

FUNC key configuration

Press the **CONFIG** key and then the **FUNC** key to display a menu that allows you to scroll through the available DMM functions. Turn the navigation wheel or press the **CURSOR** keys to scroll through available functions. Press the navigation wheel or the **ENTER** key to make the displayed function active when it is highlighted and blinking. While in the configuration mode of the **FUNC** key, the function takes effect for the highlighted function only when the **ENTER** key is pressed (the function does not change while scrolling).

INSERT key

Press the **INSERT** key to append the present channels to the scan list.

LIMIT key

Press the **LIMIT** key to cycle through the four combinations of limit state settings:

- Limit1 and Limit2 off
- Limit1 on and Limit2 off
- Limit1 off and Limit2 on
- Limit1 and Limit2 on

LIMIT key configuration

Pressing the **CONFIG** key and then the **LIMIT** key opens the LIMIT menu. Select LIMIT 1 or LIMIT 2 to open the desired LIMIT 1 or LIMIT 2 menu.

These menus contain the following items:

- **ENABLE:** Enables or disables limit testing. Related ICL command: [dmm.limit\[Y\].enable](#) (on page 13-142).
- **CLEAR:** Clears the test results of the limit. Related ICL command: [dmm.limit\[Y\].clear\(\)](#) (on page 13-141).
- **AUTOCLEAR:** Indicates if the limit should be cleared automatically or not. Related ICL command: [dmm.limit\[Y\].autoclear](#) (on page 13-141).
- **LOWVAL:** Sets the low limit value. Related ICL command: [dmm.limit\[Y\].low.value](#) (on page 13-144).
- **LOWFAIL:** Queries for the low test results of the limit. Related ICL command: [dmm.limit\[Y\].low.fail](#) (on page 13-143).
- **HIGHVAL:** Sets the high limit value. Related ICL command: [dmm.limit\[Y\].high.value](#) (on page 13-143).
- **HIGHFAIL:** Queries for the high test results of limit. Related ICL command: [dmm.limit\[Y\].high.fail](#) (on page 13-142).

LOAD key

Press the **LOAD** key to load scripts along with the Lua chunks added with [display.loadmenu.add\(\)](#) (on page 13-100) for execution. The LOAD TEST menu is displayed.

The LOAD TEST menu contains the following items:

- **USER:** Provides access to Lua chunks specified by [display.loadmenu.add\(\)](#) (on page 13-100) (not scripts).
- **SCRIPTS:** Provides access to scripts created by the user. The scripts can be directly executed.

MENU key

Press the **MENU** key to open the MAIN menu.

The MAIN menu contains the following items:

- **SCRIPT:** Opens the SCRIPT menu that contains LOAD and SAVE menu items.
- **SETUP:** Opens the SETUP menu that contains SAVE, RECALL, POWERON, and RESET menu items.
- **GPIB:** Opens the GPIB menu that contains ADDRESS and ENABLE menu items.
- **LAN:** Opens the LAN menu that contains STATUS, CONFIG, APPLY, RESET, and ENABLE menu items.
- **TSPLINK:** Opens the TSPLINK menu that contains NODE and RESET menu items.
- **UPGRADE:** Upgrades the firmware on the unit and installed cards (see Upgrade procedure using USB flash drive). This menu includes three options (YES, NO, and PREVIOUS).
- **CHANNEL:** Opens the CONNECT menu that allows you to select a rule (BBM, MBB, or OFF), or to connect sequentially (ON or OFF setting). Related ICL commands: [channel.connectrule](#) (on page 13-39) and [channel.connectsequential](#) (on page 13-40).
- **DISPLAY:** Opens the DISPLAY menu. Select the **TEST** item to open the DISPLAY TESTS menu, which contains KEYS and DISPLAY-PATTERNS menu items. Use **KEYS** to verify the operation of the keys. Use **DISPLAY-PATTERNS** to verify each segment of the display.
- **DIGIO:** Opens the DIGIO I/O menu that is used to set DIGIO-OUTPUT and WRITE-PROTECT menu items.
- **BEEPER:** Enables or disables the beeper, along with selection KEYCLICK option.
- **SYSTEM-INFO:** Opens the SYSTEM INFORMATION menu that can query FIRMWARE, SERIAL#, and CAL information.

PATT key

Press the **PATT** key to open the PATTERN ACTION menu.

- If you press the **PATT** key, but no patterns have been created or if the unit is powered up with the factory default settings, the only option that is displayed is **CREATE**.
- If you press the **PATT** key after creating channel patterns, the name of an existing pattern blinks, and you are in pattern selection mode. Use the **CURSOR** keys or navigation wheel to scroll through the available patterns, and press **ENTER** to select the one you want to use. The selected pattern is used with the **OPEN** and **CLOSE** action keys, among others.

The PATTERN ACTION menu contains the following items:

- **OPEN:** Opens the specified channel pattern for switching aspects. Related ICL command: [channel.open\(\)](#) (on page 13-59).
- **CLOSE:** Closes the specified channel pattern. These closures are appended to the already closed channels. Related ICL command: [channel.close\(\)](#) (on page 13-37).
- **EXCLOSE:** Closes the specified channel pattern so that the channels associated with the pattern are exclusively closed. Related ICL command: [channel.exclusiveclose\(\)](#) (on page 13-41).
- **EXSLOTCLOSE:** Exclusively closes specified channels in the channel pattern image for the specified slots. Related ICL command: [channel.exclusiveslotclose\(\)](#) (on page 13-43).
- **CREATE:** Creates a channel pattern from a snapshot and associates it with the specified name. From the front panel, only the first 12 characters of the name are visible. If no patterns exist in the system when the **PATT** key is pressed, **CREATE** is the only menu item that is displayed. Related ICL command: [channel.pattern.snapshot\(\)](#) (on page 13-66).
- **VIEW:** Shows the channels associated with the pattern. Related ICL command: [channel.pattern.getimage\(\)](#) (on page 13-62).
- **DELETE:** Deletes a channel pattern. Related ICL command: [channel.pattern.delete\(\)](#) (on page 13-62).
- **RESET:** Resets the channels representing the image of the selected channel pattern to the factory default settings. Also, the pattern is deleted because resetting a channel causes any patterns that contain a channel being reset to be deleted. Related ICL command: [channel.reset\(\)](#) (on page 13-68).

NOTE **CREATE** is the only item that is displayed unless a pattern has been selected.

PATT key configuration

Press the **CONFIG** key and then the **PATT** key to open the PATTERN ATTRibute menu.

The PATTERN ATTRibute menu contains the following item:

- **DMM_CONFIG**: Sets the DMM configuration associated with the specified channel pattern. Use the navigation wheel to scroll through the available DMM configurations. Related ICL command: [dmm.setconfig\(\)](#) (on page 13-168).

REL key

Press the **REL** key to enable and disable relative for the selected function. When enabled, the REL annunciator is lit. See [Relative](#) (on page 6-1).

REL key configuration

Press the **CONFIG** key and then the **REL** key to open the RELATIVE OFFSET menu.

The RELATIVE OFFSET menu contains the following menu items:

- **ACQUIRE**: Acquires an internal measurement to store as the REL level value. Related ICL command: [dmm.rel.acquire\(\)](#) (on page 13-158).
- **ENABLE**: Enables or disables relative measurement control for the DMM. Related ICL command: [dmm.rel.enable](#) (on page 13-159).
- **LEVEL**: Sets a specific offset value to use for relative measurements for the DMM. Related ICL command: [dmm.rel.level](#) (on page 13-160).

RUN key

Press the **RUN** key to run the last selected script or load menu item.

SCAN key

If the scan list is present, press the **SCAN** key to open the SCAN ACTION menu.

The SCAN ACTION menu contains the following items:

NOTE Use the **INSERT** key to create and add the present active channel to the scan list.

- **BACKGROUND:** Runs the scan. Related ICL command: [scan.background\(\)](#) (on page 13-232).
- **CREATE:** Displays following message: Use <INSERT> key.
- **LIST:** Displays the scan list (turn the navigation wheel to scroll). Related ICL command: [scan.list\(\)](#) (on page 13-237).
- **CLEAR:** Clears the scan list. Related ICL command (when sent with an empty string): [scan.create\(\)](#) (on page 13-234).
- **RESET:** Resets the scan settings to factory default values. Related ICL command: [scan.reset\(\)](#) (on page 13-242).

SCAN key configuration

Press the **CONFIG** key and then the **SCAN** key to open the SCAN ATTR menu.

The SCAN ATTR menu contains the following items:

- **ADD:** Instructs how to add an additional list of channels and/or channel patterns to scan. When you select **ADD** from the SCAN ATTR menu, "Use <INSERT> key" is displayed for a few seconds before going back to the SCAN ATTR menu options. To add items to an existing scan list, press **INSERT**.

NOTE Press the **INSERT** key when you are not in the SCAN ATTR menu on the MAIN display.

- **BYPASS:** Enables or disables bypassing the first item in the scan. Related ICL command: [scan.bypass](#) (on page 13-233).

- **MODE:** Sets the `scan.mode` value to one of the following:
 - OPEN_ALL, which is equivalent to `scan.MODE_OPEN_ALL` or 0 (default setting)
 - OPEN_SELECT, which is equivalent to `scan.MODE_OPEN_SELECTIVE` or 1
 - FIXED_ABR, which is equivalent to `scan.MODE_FIXED_ABR` or 2Related ICL command: [scan.mode\(\)](#) (on page 13-239)
- **MEAS_CNT:** Sets the measure count value. Related ICL command: [scan.measurecount](#) (on page 13-238)
- **SCAN_CNT:** Sets the scan count value. Related ICL command: [scan.scancount](#) (on page 13-242)

SLOT key

Press the **SLOT** key to display information about the installed card(s) and the main system. The information that is displayed includes firmware revisions, model names, and model numbers. After pressing this key, scroll through all available instruments, including the internal DMM (if installed), using the **CURSOR** keys, navigation wheel, or multiple presses of the **SLOT** key.

TRIG key

Press the **TRIG** key to trigger a measurement equivalent to the [dmm.measure\(\)](#) (on page 13-150) command. If the **TRIG** key is held for more than two seconds, the unit will go into continuous trigger mode and take measurements every .25 seconds (if possible, as defined by DMM attributes). Press **TRIG** or **EXIT** to stop continuous trigger mode.

The **TRIG** key can also be tied to the system trigger model and event system (see [Trigger model](#) (on page 8-4)).

Range keys, cursor keys, and navigation wheel

AUTO key

Press the **AUTO** key to enable or disable autorange for the selected function. The AUTO annunciator lights when enabled.

CURSOR keys

Press the **< CURSOR >** keys in a menu to control the cursor position when making selections or changing values.

Navigation wheel

Turn the navigation wheel to scroll to the desired menu option or to change the value of the selected numeric parameter. Pressing the navigation wheel has the same functionality as pressing the **ENTER** key. See [ENTER key](#) (on page 4-25) for more information.

When changing a multiple character value, such as an IP address or channel pattern name, press the navigation wheel to enter edit mode, rotate the navigation wheel to change the characters value as desired, but do not leave edit mode. Use the **CURSOR** keys to scroll to the other characters and use the navigation wheel to change their value as needed. Press the **ENTER** key when finished changing all the characters.

RANGE keys

Press the **RANGE** keys \blacktriangle \blacktriangledown to select the next higher or lower measurement range on the measurement display for the selected function.

If the Series 3700 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. You can also use these keys when entering a range value from the front panel. For details, see [Auto ranging over the front panel](#) (on page 5-3).

If you select a range of channels, that range must stop when the channel type changes. Therefore, you can never select a range of channels which includes different channel types.

For more information, see [Range](#) (on page 5-1).

Action keys

CLOSE key

Press the **CLOSE** key to close specified channels or channel patterns.

OPEN ALL key

Press the **OPEN ALL** key to open all closed channels.

OPEN key

Press the **OPEN** key to open selected channels or channel patterns.

RATE key

Press the **RATE** key to set the measurement speed (fast, medium, or slow) for the active or selected function. For more information, see [Rate](#) (on page 5-5).

RECall key

Press the **RECall** key to display stored readings and buffer statistics for selected reading buffer. Use the **< CURSOR >** keys or turn the navigation wheel to navigate through the buffer. For more information, see [Recalling readings](#) (on page 7-5).

STEP key

Press the **STEP** key to step through the defined scan list, where each press results in one scan step.

NOTE You cannot use an external trigger event, like digital I/O, for the channel stimulus setting of the trigger model when using the STEP key. For more information, see [Scanning](#) (on page 7-1) and [Trigger model](#) (on page 8-4).

STORE key

Press the **STORE** key to open the RD BUFF ACTION menu or <selected buffer name> menu. For more information, see [Buffer: Data Storage and Retrieval](#) (on page 7-1).

The **RD BUFF ACTION** menu contains the following items:

- **CREATE:** Allows creation of a reading buffer. When a new buffer is created, you can enter the name and set the number of readings to store. The new buffer is created with append mode ON and is automatically selected for front panel use (store readings, clear, delete, save, and so on). Related Instrument Control Library (ICL) command: [dmm.makebuffer\(\)](#) (on page 7-8).
- **SELECT:** Allows you to select a previously created reading buffer, which you can use to store readings taken on the front panel.
- **CLEAR:** Removes readings from a selected buffer.
- **SAVE:** Allows you to save a selected reading buffer to a USB flash drive (the flash drive must be installed and have enough available memory).
- **DELETE:** Lets you delete a selected reading buffer from the system. All data associated with the deleted buffer will be lost. This is equivalent to setting the reading buffer variable name to `nil` over the bus.

STORE key configuration

With a buffer selected, press the **CONFIG** key and then the **STORE** key to open the RD BUFFER ATTR menu.

This menu contains the following menu items:

- **CAPACITY:** Displays the maximum number of readings that can be stored.
- **COUNT:** Displays the actual number of readings that have been stored.
- **APPEND:** Indicates the append mode setting of the reading buffer. For buffers created on the front panel or web, this defaults to ON or enabled. For buffers created over the bus, the default is OFF or disabled.

Range, Digits, Rate, Bandwidth, and Filter

In this section:

Range	5-1
Digits ICL programming	5-4
Rate	5-5
Bandwidth	5-7
Filter	5-8

Range

The range setting is "remembered" by each measurement function. Selecting a function returns the instrument to the last range setting for that function. You cannot select a range that includes different channel types.

NOTE A power cycle or an instrument will clear "remembered" ranges.

Measurement ranges and maximum readings

The range that is selected affects both measurement accuracy as well as the maximum measurable level. Input values that exceed the maximum readings cause an "Overflow" message to be displayed.

Function	Ranges	Maximum reading
DCV (DC voltage)	100mV, 1V, 10V, 100V, 300V	± 303V
ACV (AC voltage)	100mV, 1V, 10V, 100V, 300V	± 303V
DCI (DC current)	10µA, 100µA, 1mA, 10mA, 100mA, 1A	± 3.1A
ACI (AC current)	1mA, 10mA, 100mA, 1A	± 3.1A
Ω2 (2-wire ohm)	10Ω, 100Ω, 1kΩ, 10kΩ, 100kΩ, 1MΩ, 10MΩ, 100MΩ	120MΩ
Ω4 (4-wire ohm)	1Ω, 10Ω, 100Ω, 1kΩ, 10kΩ, 100kΩ, 1MΩ, 10MΩ, 100MΩ	120MΩ
Ω4 OC (4-wire ohm offset compensated)	1Ω, 10Ω, 100Ω, 1kΩ, 10kΩ	12kΩ
Ω4 DRY+ (4-wire ohm dry circuit)	1Ω, 10Ω, 100Ω, 1kΩ, 10kΩ	2.4kΩ

Function	Ranges	Maximum reading
TMP (temperature)	-200°C to 1820°C	Sensor dependent
FREQ (frequency)	100mV, 1V, 10V, 100V, 300V	3Hz to 500kHz
PER (period)	100mV, 1V, 10V, 100V, 300V	2μs to 333ms
CNT (continuity)	1kΩ Threshold adjustable 1Ω to 1000Ω	
CSΩ (common-side ohm)	1Ω, 10Ω, 100Ω, 1kΩ, 10kΩ, 100kΩ, 1MΩ, 10MΩ, 100MΩ	120MΩ

Temperature

There is no range selection for temperature measurements, which are performed on a single fixed range. Depending on the sensor, the maximum temperature readings range from -200°C to 1820°C.

Manual range keys

To change range, press the **RANGE ▲** or **▼** key. The instrument changes one range value of the active function per key press. The selected range is displayed in the attribute list on the second line of the front panel display.

NOTE The manual range keys have no effect on temperature measurements.

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

Auto ranging over the front panel

To enable auto range, press the **AUTO** key. The AUTO indicator turns on when auto ranging is selected. While auto ranging is enabled, the instrument automatically selects the best range to measure the applied signal. Auto ranging should not be used when optimum speed is required.

NOTE The **AUTO** key has no effect on temperature measurements.

Up-ranging occurs at 120% of range. The Series 3700 will down-range when the reading is <10% of nominal range.

To disable auto ranging, press the **AUTO** key. This will leave the instrument on the present range.

You can also disable auto ranging by pressing the ▲ or ▼ key. However, this may cause a range change.

Scanning

Each channel of scan configuration can be associated with a unique digital multimeter (DMM) configuration (which includes a range setting). When a scan completes, the DMM remains in the configuration associated with the last completed measurement step. For remote programming, the `<ch_list>` parameter is used to configure channels for a scan (see [*dmm.configure.set\(\)*](#) (on page 13-129), [*dmm.setconfig\(\)*](#) (on page 13-168), [*scan.create\(\)*](#) (on page 13-234), and [*scan.add\(\)*](#) (on page 13-230) commands, and the [Scanning](#) (on page 7-1) section for more details, including front panel operation).

Range remote programming (ICL)

Instrument control library (ICL) commands are sent to the Series 3700 when controlling it over the bus. See [Instrument Control Library \(ICL\)](#) (on page 12-1) for detailed information.

Selecting a manual range

The range is selected by specifying the expected reading as an absolute value using the `<n>` parameter for the `dmm.range` (on page 13-156) command. The Series 3700 will then go to the most sensitive range for that expected reading. For example, if you expect a reading of approximately 3V, let the parameter `<n>` equal 3 (`dmm.range = 3`) to select the 10V range.

Selecting an auto range

Auto range is enabled by setting the `dmm.autorange` attribute (see [dmm.autorange](#) (on page 13-113)) to either `dmm.ON` or 1. When auto range is enabled, the range is changed automatically for the selected range value. When auto range is disabled, the instrument remains at the selected range. To disable auto range, either set the `dmm.autorange` attribute to `dmm.OFF` or 0, or send a valid `dmm.range` attribute (see [dmm.range](#) (on page 13-156)).

Digits ICL programming

Set the `dmm.displaydigits` (on page 13-132) attribute to change the display resolution for the Series 3700 from 3½ to 7½ digits. Each mainframe input function can have its own unique digits setting. Digits has no effect on the remote reading format. The number of displayed digits does not affect accuracy or speed.

Scanning

When a scan is configured, each channel can have its own unique digits setting.

Setting digits

Even though the parameters for the `dmm.displaydigits` attribute (see [dmm.displaydigits](#) (on page 13-132) for remote control or DIGITS under the function attribute menu on front panel) are expressed as integers (3 to 7), you can specify resolution using a real number. For example, to select 3½ digit resolution, let `<n> = 3.5`. Internally, the instrument rounds the entered parameter value to the nearest integer. Each mainframe input function can have its own unique digits setting.

Rate

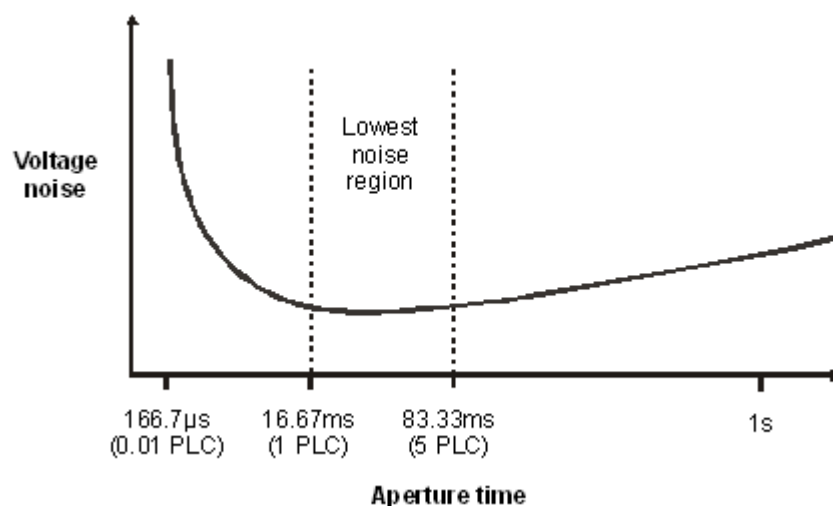
The **RATE** key sets the integration time (measurement speed) of the A/D converter. This controls how long the input signal is measured (also known as aperture). The integration time affects the amount of reading noise, as well as the ultimate reading rate of the instrument.

The integration time is specified in parameters based on a 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 (0.1 PLC using the front panel **RATE** key, or 0.0005 PLC from the bus or through the **DMM > CONFIG NPLC** menu) results in increased reading noise and fewer usable digits, while the slowest integration time (5 PLC using the front panel **RATE** key, or 15 PLC from the bus or through the **DMM > CONFIG NPLC** menu) provides the best common-mode and normal-mode rejection. In-between settings are a compromise between speed and noise. To set the NPLC from 0.0005 to 15 on the front panel, press the **CONFIG** key, and then press the **DMM** key to open the function attribute menu. From the function attribute menu, select **NPLC** to dial in a specific value for NPLC.

The Series 3700 has a parabola-like shape for its speed versus noise characteristics. The Series 3700 is optimized for the 1 PLC to 5 PLC reading rate. At these rates (lowest noise region in graph), the Series 3700 will make corrections for its own internal drift and will still be fast enough to settle a step response <100ms.

Figure 5-1: Speed versus noise characteristics



The front panel RATE settings for all but the AC functions are explained as follows:

- FAST sets integration time to 0.1 PLC. Use FAST if speed is of primary importance (at the expense of increased reading noise and fewer usable digits).
- MEDium sets integration time to 1 PLC. Use MEDium when a compromise between noise performance and speed is acceptable.
- SLOW sets integration time to 5 PLC. SLOW provides better noise performance at the expense of speed.

For the AC functions (ACV, ACV dB, and ACI), the **RATE** key sets integration time and bandwidth. As listed in the following table, FAST sets NPLC to 1, while the MEDium and SLOW NPLC settings are ignored.

Function	Slow	Medium	Fast
DCV, DCI	NPLC=5	NPLC=1	NPLC=0.1
ACV, ACI	NPLC=X, BW=3	NPLC=X, BW=30	NPLC=1, BW=300
Ω 2, Ω 4, CS Ω	NPLC=5	NPLC=1	NPLC=0.1
FREQ, PERIOD	APER=0.250s	APER=0.1s	APER=0.01s
Continuity	X	X	NPLC=0.006
NOTES: NPLC = Number of power line cycles. BW = Bandwidth (in Hz). APER = Aperture in seconds. X = Setting ignored (fixed NPLC).			

You can use unique rate settings for each function when using the front panel or the remote interface.

NOTE Rate cannot be set for continuity; it is fixed at 0.006PLC.

Setting Rate from the front panel

The **RATE** key is used to set measurement speed from the front panel. Press the **RATE** key until the desired speed message is displayed. The second line of the display will contain the NPLC setting. Alternatively, use the NPLC option under the function attribute menu (press **CONFIG > DMM** keys to display).

NOTE The Series 3700 uses internal references to calculate an accurate and stable reading. When the NPLC setting is changed, each reference will be automatically updated to the new NPLC setting before a reading is generated. Therefore, frequent NPLC setting changes can result in slower measurement speed.

Setting measurement speed from a remote interface

Use the `dmm.aperture` (on page 13-110) or `dmm.nplc` (on page 13-152) command to set the measurement speed (integration time) over the bus.

Bandwidth

There are three bandwidth settings for ACV and ACI measurements. The **RATE** setting determines the bandwidth setting as follows:

- **SLOW**: 3Hz to 30Hz
- **MEDium**: 30Hz to 300Hz
- **FAST**: 300Hz to 300MHz

When the **SLOW** bandwidth (3Hz to 30Hz) is chosen, the signal goes through an analog root-mean-square (RMS) converter. The output of the RMS converter goes to a fast (1kHz) sampling A/D and the RMS value is calculated from 1200 digitized samples (1.2s).

When the **MEDium** bandwidth (30Hz to 300Hz) is chosen, the same circuit is used. However, only 120 samples (120ms) are needed for an accurate calculation because the analog RMS converter has turned most of the signal to DC.

In the FAST bandwidth (300Hz to 300kHz), the output of the analog RMS converter (nearly pure DC at these frequencies) is measured at 1 PLC (16.6ms). For remote programming, the integration rate can be set from 0.0005PLC to 12PLC or 15PLC.

To achieve the best accuracy for ACV and ACI measurements, use the bandwidth setting that best reflects the frequency of the input signal. For example, if the input signal is 40Hz, then a bandwidth setting of 30 should be used.

NOTE A rate command (`dmm.nplc` (on page 13-152) or `dmm.aperture` (on page 13-110)) for ACV and ACI is only valid if the bandwidth for that AC function is set to 300 (300Hz to 300kHz). Bandwidth is set using the `dmm.detectorbandwidth` (on page 13-132) ICL command or the DETECTBW menu option under the function's attribute menu).

Filter

The digital filter is used to stabilize noisy measurements. The displayed, stored, or transmitted reading is a windowed-average of a number of reading conversions (from 1 to 100).

The filter setup is retained and can be unique for each measurement function (DCV, ACV, DCI, ACI, $\Omega 2$, $\Omega 4$, $CS\Omega$, and TEMP). When you select a function, the instrument will return to the last filter setup for that function.

NOTE The various instrument operations, including Filter, are performed on the input signal in a specific, predetermined order. For example, if both REL and MXB (a math operation) are enabled, the REL operation will always be performed before MXB.

Filter characteristics

In general, the digital filter places a specified number of A/D conversions ("Filter Count") into a memory stack. These A/D conversions must occur consecutively within a selected reading window ("Filter Window"). The readings in the stack are then averaged to yield a single filtered reading. The stack can be filled using the moving or repeating average filters. Details about digital filter characteristics are provided in the following paragraphs.

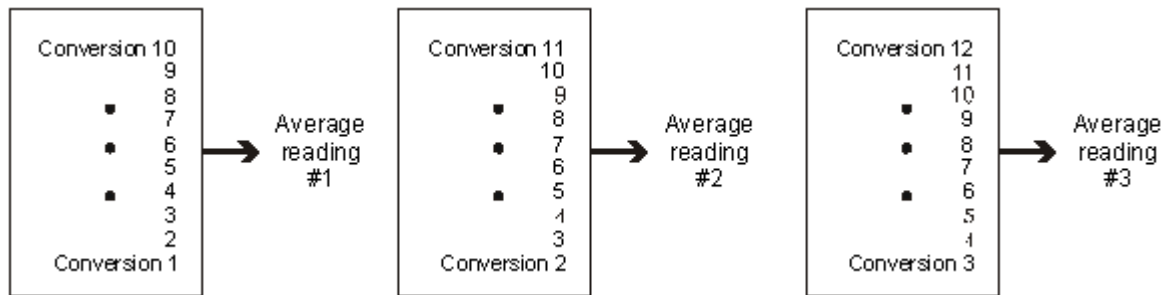
Digital filter types

There are two digital filter types: moving and repeating.

Moving average filter

The moving average filter uses a first-in first-out stack, where the newest reading conversion replaces the oldest. An average of the stacked reading conversions yields a filtered reading. After the specified number of reading conversions ("Filter count") fill the stack, the moving filter gives a new reading for every new conversion.

Figure 5-2: Moving average filter

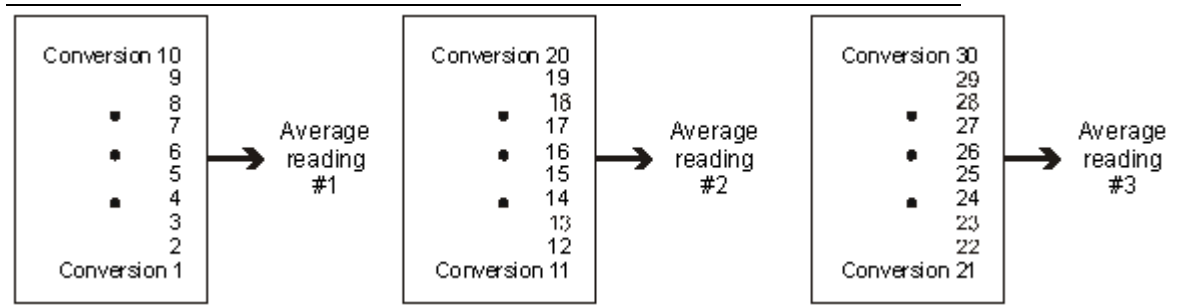


Repeating average filter

The repeating filter takes a specified number of conversions, averages them, and yields a filtered reading. It then clears its stack and starts over. This setting is useful when scanning because readings for other channels are not averaged with the present channel. The stack is then cleared and the process starts over.

NOTE The moving filter cannot be used when scanning (see [Scanning](#) (on page 7-1)). If a scan channel is set up to use the moving filter, the filter will not turn on.

Figure 5-3: Repeating average filter



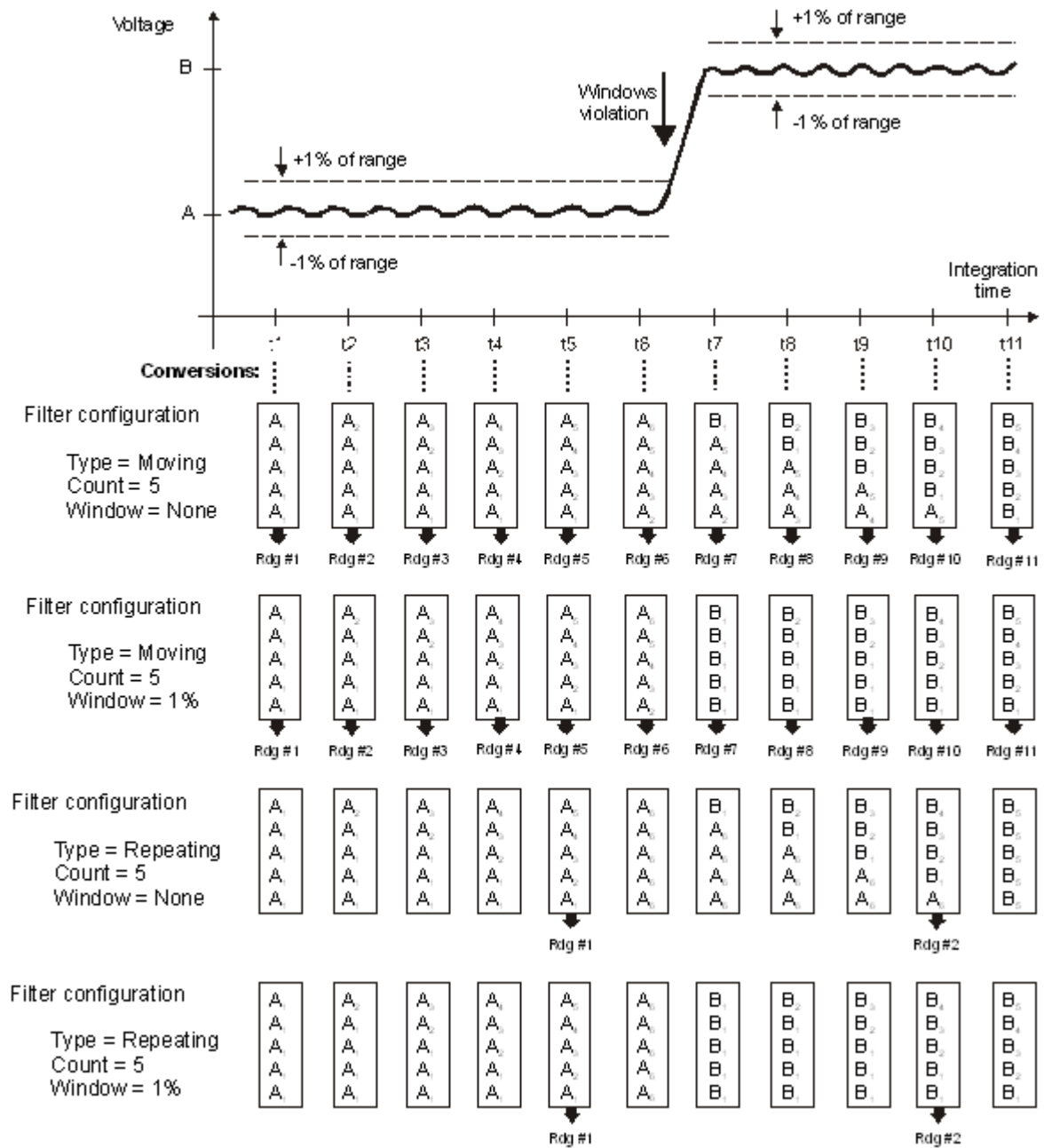
Digital filter window

The digital filter uses a "noise" window to control the filter threshold. As long as the input signal remains within the selected window, A/D conversions continue to be placed in the stack. If the signal changes to a value outside the window, the filter resets and starts processing again, starting with a new initial conversion value from the A/D converter.

The noise window, which is expressed as a percentage of range (or maximum temperature reading), allows a faster response time to large signal step changes (for example, scanned readings). A reading conversion outside the plus or minus noise window fills the filter stack immediately.

If the noise does not exceed the selected window, the reading is based on the average of the reading conversions. If the noise does exceed the selected window, the reading is a single reading conversion and new averaging starts from this point. The noise window for the two filter types are compared in the filter window below.

Figure 5-4: Filter window



For both front panel and remote programming, the window can be set to any value from 0.0% to 10%, where 0.0% represents no window being applied.

For voltage, current, and resistance, the filter window is expressed as a percent of range. For example, on the 10V range, a 10% window means that the filter window is $\pm 1V$.

For temperature, the filter window is expressed as a percent of the maximum temperature reading. The maximum temperature depends on which thermocouple is being used. For example, for a Type J thermocouple, the maximum reading is $760^{\circ}C$; a 10% window means that the filter window is $\pm 76^{\circ}C$. For temperatures below $0^{\circ}C$, the overflow point is $-200^{\circ}C$, so a 10% filter window is $\pm 20^{\circ}C$. If using $^{\circ}F$ units, a 20% filter window is calculated as follows: $9/5 \times 20 = 36$. The filter window for the 20% window is $\pm 36^{\circ}C$.

Relative, Math, and dB

In this section:

Relative	6-1
Math calculations	6-3
dB commands	6-10

Relative

Use the REL (relative) function to null offsets or subtract a baseline reading from present and future readings. When REL is enabled, subsequent readings will be equal to the difference between the actual input value and the REL value.

You can define a REL value for each function. Once a REL value is established for a measurement function, the value is the same for all ranges. For example, if 50V is set as a REL value on the 100V range, the REL is also 50V on the 1000V, 10V, 1V, and 100mV ranges.

Therefore, when you perform a zero correction, the displayed offset becomes the reference value. Subtracting the offset from the actual input zeros the display, as follows:

$$\text{Actual input} - \text{Reference} = \text{Displayed reading}$$

A REL value can be as large as the highest range.

Selecting a range that cannot accommodate the REL value does not cause an overflow condition, but it also does not increase the maximum allowable input for that range. For example, on the 10V range, the Series 3700 still overflows for a 12V input.

NOTE The various instrument operations, including REL, are performed on the input signal in a specific, predetermined order. For example, if both REL and MXB (a math operation) are enabled, the REL operation will always be performed before MXB.

Basic front panel REL procedure

1. Select the desired measurement function and an appropriate range setting.
2. Apply the signal you want to REL to a switching channel input or to the Series 3700 inputs.
3. If you are using a switching module, close the input channel. (see [Operation keys](#) (on page 4-17) for basic information about the front panel graphical user interface).
4. Press the **REL** key to acquire the REL value. The REL annunciator will turn on, but the displayed value will not become zero until a new reading is triggered.
5. Apply the signal to be measured. The relative value will be subtracted from the next reading that is triggered.

Pressing **REL** a second time disables the REL function. You also have the option to manually enter a REL value rather than acquiring a value from an input signal. To do this, select **REL > LEVEL** from the front panel and enter the value. You will still need to enable REL by selecting the **ENABLE** menu item. You can also perform a REL acquire from this menu. Note that pressing the **REL** key is equivalent to doing an acquire followed by enabling REL.

NOTE You can perform the equivalent of REL manually by using the *mX+b* (on page 6-4) math function. Set M for 1 and B for any value you want.

REL remote operation

The `dmm.rel.level()` command specifies the REL value (for the active function only), while the `dmm.rel.acquire()` command uses the input signal as the REL value (again, for the active function only). The `dmm.rel.acquire()` command is typically used to zero the display. For example, if the instrument is displaying a 1 μ V offset, sending `dmm.rel.acquire()` and enabling REL (`dmm.rel.enable = dmm.ON`) zeros the display.

The following command sequence is equivalent to pressing the front panel **REL** key:

```
dmm.rel.acquire()  
dmm.rel.enable=dmm.ON
```

To manually set a REL value of 1.5 μ V, use this command sequence:

```
dmm.rel.level=1.5e-6  
dmm.rel.enable=dmm.ON
```

For example, if the instrument is on the DCV function, the `dmm.rel.acquire()` command is applicable to DCV measurements.

Scanning

When a scan is configured, each channel can have its own unique REL value. For remote programming, the `<ch_list>` parameter is used to configure channels for a scan.

For example:

To attach a 1 μ V REL level to a desired configuration, send the following commands:

```
-- Select DC volts function.
dmm.func = 'dcvolts'

-- Reset DC volts only.
dmm.reset('active')

-- Set the rel level.
dmm.rel.level=1e-6

-- Enable REL.
dmm.rel.enable = dmm.ON

-- Call the configuration myconfig.
dmm.configure.set('myconfig')

-- Set Channels 1001 to 1030 to use myconfig configuration.
dmm.setconfig('1001:1030', 'myconfig')

-- Create scan list of channels 1001 to 1030 using
  myconfig.
scan.create('1001:1030")
```

Math calculations

The Series 3700 has three built-in math calculations that are accessed from the MATH menu: $mX+b$, percent, and reciprocal ($1/X$). The settings shown in the menu tree are the factory default settings.

NOTE The various instrument operations, including Math, are performed on the input signal in a specific, predetermined order. For example, if both REL and MXB (a math operation) are enabled, the REL operation will always be performed before MXB.

mX+b

This math operation lets you manipulate normal display readings (X) mathematically according to the following calculation:

$$Y = mX + b$$

Where:

- **X** is the normal display reading.
- **m** and **b** are the user-entered constants for scale factor and offset.
- **Y** is the displayed result.

Use `dmm.math.mxb.bfactor` (on page 13-148) and `dmm.math.mxb.mfactor` (on page 13-149) to set the b and m factor for mX+b.

Once all settings are configured, use `dmm.math.enable = dmm.ON` to enable math operation.

NOTE The REL'ed reading of the input signal (if using REL) is used by the mX+b calculation.

mX+b REL

Use the mX+b function to manually establish a REL value. To do this, set the scale factor (M) to 1 and set the offset (b) to the REL value. Each subsequent reading will be the difference between the actual input and the REL value (offset).

Setting mX+b units

The attribute for `dmm.math.mxb.units` (on page 13-149) must be one character enclosed in single or double quotes. It can be any letter of the alphabet, the degrees symbol (°), or the ohms symbol (Ω).

To set mX+B units from the front panel:

NOTE The following procedure sets MXBUNITS. You can change the other MATH menu options (BFACTOR and MFACTOR) by changing the b and m values.

1. Press the **CONFIG** key.
2. Press the **DMM** key.
3. Turn the navigation wheel to highlight the **MATH** menu item.
4. With **MATH** highlighted, press the **ENTER** key. The MATH MENU opens.
5. Select the **MXBUNITS** menu item.
6. With **MXBUNITS** highlighted, press the **ENTER** key.
7. Press the navigation wheel to enter EDIT mode.
8. Scroll until the desired character is displayed, and then press the **ENTER** key. The MATH MENU will open.
9. From the MATH MENU, turn the navigation wheel to highlight and select the **ENABLE** menu item.
10. Select **ON** and press the **ENTER** key.
11. Press the **EXIT** key twice to return to the main display.

To set mX=B units from a remote interface:

The ohms symbol (Ω), the micro symbol (μ), and the degrees symbol ($^\circ$) are not ASCII characters and must be substituted with the ']', '[' and '\ ' characters. Valid characters are therefore from A to Z, ']' for ohms, '[' for microvolts, and '\ ' for degrees.

To use the ohms symbol (Ω) as units designator:

```
value = ']'  
dmm.math.mxb.units = value
```

To use the micro symbol (μ) as units designator:

```
value = '['  
dmm.math.mxb.units = value
```

To use the degrees symbol ($^\circ$) as units designator:

```
value = '\\ '  
dmm.math.mxb.units = value
```

NOTE When sending mxb units remotely, to embed a '\ ' into a string, precede the '\ ' with an additional '\ ' (see the previous paragraph).

Percent

This math function determines percent deviation from a specified reference value. The percent calculation is performed as follows:

$$\text{Percent} = \left(\frac{\text{input} - \text{reference}}{\text{reference}} \right) \times 100\%$$

Where:

Input: The normal measurement (if using REL, it will be the REL'ed value)

Reference: The user-entered constant (`dmm.math.percent`)

Percent: The result

NOTE The REL'ed reading of the input signal (if using REL) is used by the percent calculation.

NOTE The result of the percent calculation is positive when the input exceeds the reference and negative when the input is less than the reference. The result of the percent calculation may be displayed in exponential notation. For example, a displayed reading of +2.500E+03% is equivalent to 2500% (2.5K%).

The `dmm.math.percent` attribute (see [dmm.math.percent](#) (on page 13-150)) specifies the reference value for the percent calculation, while the `dmm.rel.acquire` function (see [dmm.rel.acquire\(\)](#) (on page 13-158)) uses the input signal as the reference value.

The `acquire` function triggers a single reading and uses the result as the new REL value. When a value is set using [dmm.math.percent](#) (on page 13-150), the [dmm.math.percent](#) (on page 13-150) query command returns the programmed value. When reference is set using [dmm.rel.acquire\(\)](#) (on page 13-158), the [dmm.math.percent](#) (on page 13-150) query command returns the acquired reference value.

To set a percent value from a remote interface, send the following commands:

```
-- Set percent to 5
dmm.math.percent = 5

-- Sends 5 to the PC for display
print(dmm.math.percent)
```

To set a percent value on the front panel:

1. Open the function attribute menu:
 - Press the **CONFIG** key.
 - Press the **DMM** key.
2. Turn the navigation wheel to highlight the **MATH** menu item.
3. With **MATH** highlighted, press the **ENTER** key. The MATH MENU opens.
4. Select the **PERCENT** menu item.
5. Press the **ENTER** key to enter edit mode.
6. Turn the navigation wheel to edit the value.
7. Once the desired value is displayed, press the **ENTER** key. The MATH MENU opens.
8. From the MATH MENU, turn the navigation wheel to highlight and select the **ENABLE** menu item.
9. Select **ON** and press the **ENTER** key.
10. Press the **EXIT** key twice to return to the main display.

REL can be used to set percent (bus-only operation) as follows:

```
-- Sets percent with REL acquire value.
dmm.math.percent = dmm.rel.acquire()

-- Send the result of REL acquire to a computer.
print(dmm.math.percent)
```

Reciprocal (1/X)

The reciprocal of a reading is displayed when the reciprocal (1/X) math function is enabled:

$$dB = 20 \log \frac{V_{in}}{V_{ref}}$$

Where: X is the normal input reading

The displayed units designator for reciprocal readings is "R." You cannot change this units designator.

Example:

Assume the normal displayed reading is 002.5000 Ω . The reciprocal of resistance is conductance. When the reciprocal math function is enabled, the following conductance reading will be displayed:

0.400000 R

NOTE The result of the 1/X calculation may be displayed in exponential notation. For example, a displayed reading of +2.500E+03 R is equivalent to 2500 R (2.5K R). When using REL, the REL'ed reading of the input signal is used by the 1/X calculation.

Basic reciprocal operation

1. Select the desired measurement function.
2. Configure and enable the mX+b, percent, or reciprocal (1/X) math function as previously explained.
3. Apply the signal to be measured to a switching channel input.
4. Close the input channel. The result of the math calculation will be displayed when a reading is triggered.

Scanning

When a scan is configured, each channel can have its own unique math setup. For remote programming, the `<ch_list>` parameter is used to configure channels for a scan.

Example:

To perform the reciprocal math operation on DC volt measurements, send the following commands:

```
-- Select DC volts function.
dmm.func = 'dcvolts'

-- Reset DC volts only.
dmm.reset('active')

-- Set the math operation to be reciprocal for
  measurements.
dmm.math.format = dmm.MATH_RECIPROCAL

-- Enable the math operation for measurements.
dmm.math.enable = dmm.ON

-- Call the configuration mymath.
dmm.configure.set('mymath')

-- Set Channels 1001 to 1030 to use the mymath
  configuration.
dmm.setconfig('1001:1030', 'mymath')

-- Create scan list of channels 1001 to 1030 using mymath.
scan.create('1001:1030')
```

dB commands

Expressing DC or AC voltage in dB makes it possible to compress a large range of measurements into a much smaller scope. The relationship between dB and voltage is defined by the following equation:

$$dB = 20 \log \frac{V_{in}}{V_{ref}}$$

Where:

V_{IN} : DC or AC input signal.

V_{REF} : Specified voltage reference level.

The instrument will read 0dB when the reference voltage level is applied to the input. If a relative value is in effect when dB is selected, the value is converted to dB, and then REL is applied to dB. If REL is applied after dB has been selected, dB has REL applied to it.

NOTE The dB calculation takes the absolute value of the ratio V_{IN} / V_{REF} . The largest negative value of dB is -160dB. This will accommodate a ratio of $V_{IN} = 1\mu V$ and $V_{REF} = 1000V$.

dB configuration

You can select UNITS (V or dB) from the front panel.

To select UNITS from the front panel, while in DCV or ACV:

1. Press the **CONFIG** key.
2. Press the **DMM** key.
3. Turn the navigation wheel to scroll to the **UNITS** menu item.
4. Press the navigation wheel (or the **ENTER** key) to select.
5. Select units: V for voltage or dB for decibels.
6. Press the navigation wheel (or the **ENTER** key) to set.
7. Press the **EXIT** key to close the attribute menu.

dB scanning

Each channel in a scan may be configured to use dB. Create a configuration that has the dB enabled for units for the desired function by using the `dmm.configure.set` command. Once the configuration exists, use the `dmm.setconfig()` (on page 13-168) command to connect the configuration to the desired channels. Now the channels can be added to scanning (see `scan.create()` (on page 13-234) and `scan.add()` (on page 13-230) commands). To remotely control the units for AC and DC volts, use the `dmm.units` (on page 13-174) command.

Buffer: Data Storage and Retrieval

In this section:

Buffer overview	7-1
Front panel operation	7-2
Remote buffer operation	7-7

Buffer overview

The Keithley Instruments Series 3700 System Switch/Multimeter uses synchronous reading acquisitions to take readings for a dynamically-created reading buffer. The instrument stores the numbered readings that are acquired during the storage process. Each reading includes reading units with options that include time stamp and channel information. All routines that return measurements can return the measurements in a reading buffer. Synchronous measurements return a single value or both a single value and a reading buffer. More advanced users can access the additional information stored in the reading buffer.

You can configure single-point measurement routines to make multiple measurements where only one would ordinarily be made. Also, the measured value is not the only component of a reading. The measurement status (for example, limit or overflow) is also data associated with a particular reading.

Create and configure buffers using the front panel or through a remote interface using the Instrument Control Library (ICL) commands.

CAUTION	Once you create a reading buffer, using that buffer name for another buffer or variable will cause access to the original data to be lost.
----------------	--------------------------------------------------------------------------------------------------------------------------------------------

Reading buffer names are just like any other global variables in the system. For example, if `buf1` is a reading buffer name, then `buf1 = 5` will cause the reading buffer data currently associated with `buf1` to be lost and `buf1` to equal 5.

NOTE	The various instrument operations, including buffer operation, are performed on the input signal in a specific, predetermined order. For example, if both REL and MXB (a math operation) are enabled, the REL operation will always be performed before MXB.
-------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Front panel operation

In the following procedures, pressing in the navigation wheel (as a button) will perform the same function as pressing the **ENTER** key. Also, you can turn the navigation wheel instead of using the **< CURSOR >** keys.

Read [Creating and selecting a reading buffer](#) (on page 7-2) or [Selecting a reading buffer](#) (on page 7-3) before performing the following procedures:

- [Storing readings](#) (on page 7-3)
- [Saving readings](#) (on page 7-3)
- [Clearing readings](#) (on page 7-4)
- [Deleting a reading buffer](#) (on page 7-5)
- [Recalling readings](#) (on page 7-5)
- [Buffer configuration \(front panel\)](#) (on page 7-6)
- [Appending readings](#) (on page 7-7)

Creating and selecting a reading buffer

To create a new reading buffer that will be automatically selected after it is created:

1. Press the **STORE** key.
2. Select **CREATE** from the buffer choices and press the **ENTER** key.
3. Using the navigation wheel and the **< CURSOR >** keys, scroll through the characters, changing them until the desired name is shown.

NAME = _ _ _ _ _

4. Press the **ENTER** key. The starting name is:

f p b u f n _ _ _ _ _

Where:

f p = front panel

buf = buffer

n = number, sequentially incremented

5. Specify the number of readings to store in the buffer.
6. Press the **ENTER** key. The append attribute of this buffer is enabled (set to 1).

NOTE The newly-created buffer is automatically selected as the buffer for storing front panel readings.

Selecting a reading buffer

You can only select an existing reading buffer. If necessary, create it first. See [Creating and selecting a reading buffer](#) (on page 7-2) for more information.

NOTE When you create a new reading buffer from the front panel, it is automatically selected.

To select a reading buffer:

1. Set up the Series 3700 to take measurements.
2. Press the **STORE** key.
3. Select **SELECT** from the buffer choices and press the **ENTER** key.
4. Use the **< CURSOR >** keys to select the desired buffer.

Storing readings

Before storing readings, make sure you have selected the desired reading buffer. See [Selecting a reading buffer](#) (on page 7-3) for more information.

To store a reading, press the **TRIG** key or execute a scan. The asterisk (*) annunciator turns on, which indicates that the buffer is enabled, and turns off when storage is finished. The annunciator stays on as long as the created buffer's capacity is less than the number of readings stored.

To stop the buffer, press the **EXIT** key, or if you are taking continuous readings, press the **TRIG** key.

NOTE Stored readings are lost when the instrument is turned off. To save your stored readings, see [Saving readings](#) (on page 7-3).

Saving readings

When saving readings to a USB flash drive, you must select a non-empty reading buffer. See [Selecting a reading buffer](#) (on page 7-3) for more information.

To save readings to a USB flash drive:

1. Select a non-empty reading buffer.
2. Press the **STORE** key. The [BUFFER NAME] MENU is displayed.
3. Select the **SAVE** menu item, and press the **ENTER** key. The SAVE RD BUFFER menu is displayed.
4. Press the **ENTER** key when **USB** is highlighted.
5. Using the navigation wheel and **< CURSOR >** keys, enter the filename where the data will be saved on the installed USB flash drive. The starting name is:

<reading buffer name>_nn_ _ _

Where: nn starts at 01 and automatically increments. For example, if the selected reading buffer is `fpbuf1`, then the starting name is `fpbuf1_01_ _ _`.

6. Press the **ENTER** key to save the data to the installed USB flash drive or the **EXIT** key to cancel.

Clearing readings

When clearing readings, you must select a reading buffer. See [Selecting a reading buffer](#) (on page 7-3) for more information.

To clear readings:

1. Select a reading buffer.
2. Press the **STORE** key. The [BUFFER NAME] MENU is displayed.
3. Select the **CLEAR** menu item, and press the **ENTER** key.
4. At the prompt, select **YES** or **NO** and press the **ENTER** key.

Deleting a reading buffer

To delete a reading buffer:

1. Select the reading buffer you want to delete.
2. Press the **STORE** key. The [BUFFER NAME] MENU is displayed.
3. Select the **DELETE** menu item, and press the **ENTER** key.
4. At the prompt, select **YES** or **NO** and press the **ENTER** key.
 - If you select **YES**, the RD BUFF ACTION MENU is displayed.
 - If you select **NO**, the [BUFFER NAME] MENU is displayed.

NOTE To delete a buffer (including front panel buffers) remotely (over the bus), set the buffer's name to `nil`. For example, to delete a buffer named FPBUF1, send the command: `FPBUF1 = nil`.

Recalling readings

When recalling readings, you must select a non-empty reading buffer. See [Selecting a reading buffer](#) (on page 7-3) for more information.

Readings stored in the buffer are displayed by pressing the **REC** key. Turn the navigation wheel or use the **< CURSOR >** keys to cycle through the buffer's contents. A message is displayed if a buffer is empty.

When recalling a buffer, the display contains the following information:

- Measurement reading for each entry is at the top right.
- Buffer location number is at the bottom left.
- *Time stamp* (on page 7-5) (if used) is positioned at the bottom right.
- *Channel display* (on page 7-6) or channel pattern (if used) associated with the reading for each entry is at the top left.

Time stamp

When time stamps are enabled, they are shown in absolute time and stored as the number of seconds in Universal Coordinated Time (UTC) format. Therefore, the displayed time stamp will show month, day, and year, as well as hour, minutes, seconds, and fractional seconds.

Channel display

The returned value provides different information, based on what is opened or closed when the reading is acquired:

- If no channel or channel pattern is closed when the reading is acquired, "None" is displayed.
- If only a single channel or backplane relay is closed, the channel number is displayed (for example, 5003 or 5915).
- If a channel or backplane relay plus another backplane relay or other channel is closed, then the channel number is displayed, followed by a + sign (for example, 3005+ or 3915+). The channel is in the image unless the last close operation involved only backplane relays.
- If multiple channels and/or backplane relays are closed in a channel list, the last channel specified is stored. Channels take precedence over backplane relays when stored. However, if only multiple backplane relays are specified, then the first one is stored.
- If a channel pattern is closed, then the first eight characters of the channel pattern name are returned (for example, `mypattern1` is shown as `mypatter`).

Buffer configuration (front panel)

When configuring the buffer through the front panel, you must select a reading buffer. See [Selecting a reading buffer](#) (on page 7-3) for more information.

1. Press the **CONFIG** key.
2. Press the **STORE** key. The RD BUFFER ATTR menu opens.
3. To view the count and capacity of a selected buffer, select the **COUNT** or **CAPACITY** menu choice. To configure the buffer's append mode, select **APPEND**, then **ON** or **OFF**.

Appending readings

When the buffer append mode is disabled, the buffer is cleared (readings lost) before a new storage operation starts.

When buffer append mode is enabled, the buffer is not cleared and each subsequent storage operation appends the readings to the buffer. When the buffer is filled to capacity, the storage process stops. The readings must be cleared before the next storage operation starts.

See [dmm.appendbuffer\(\)](#) (on page 13-111) for more information.

NOTE Buffers created on the front panel have the append mode enabled by default.

Remote buffer operation

Control the Series 3700 buffer over the bus by sending ICL (Instrument Control Library) commands.

Data store (buffer) commands

The following commands are associated with data store operation:

- `dmm.makebuffer()` (on page 7-8)
- `dmm.savebuffer()` (on page 7-10)

To delete a dynamically allocated buffer, use the command `mybuffer = nil`.

Command	Description
<code>dmm.buffer.catalog()</code>	An iterator that can act on all reading buffer names in the system.
<code>dmm.buffer.info("buffer name")</code>	Returns the number of stored readings in the specified buffer, along with the overall buffer capacity. The first returned value is the stored readings number, while the second is the capacity.
<code>dmm.buffer.maxcapacity</code>	Returns the overall maximum storage capacity of all reading buffers in the system.
<code>dmm.buffer.usedcapacity</code>	Returns the sum storage capacity allocated for all currently created reading buffers in the system.

To see the current storage number and capacity of all reading buffers in the system, use the following at a Test Script Processor (TSP™) prompt or in a script:

```
for n in dmm.buffer.catalog() do stored, cap =
  dmm.buffer.info(n) print(n, 'stored = ' .. stored,
    'capacity = ' .. cap) end
```

Sample output

```
buf1      stored = 0   capacity = 1000
buf2      stored = 0   capacity = 2000
buf4      stored = 0   capacity = 4000
buf5      stored = 0   capacity = 5000
buf3      stored = 0   capacity = 3000
```

As the sample output shows, the system has five reading buffers, but currently none of them have data stored in them (`stored = 0`). The storage capacity of the buffers range from 1000 to 5000. If you send:

```
print(dmm.buffer.usedcapacity)
```

The output is `1.500000000e+004`.

Overall, the system is allocating 15000 of its max storage capacity for reading buffers.

dmm.makebuffer()	
Function	Creates a user buffer for storing readings.
Usage	<code>mybuffer = dmm.makebuffer(buffer_size)</code> buffer_size : Maximum number of readings that can be stored.
Remarks	These reading buffers are allocated dynamically. This function creates the buffers where <code>buffer_size</code> indicates the maximum number of readings the buffer can store. These buffers can be deleted by setting <code>mybuffer</code> to <code>nil</code> .
Details	<p>Once a buffer is created, the attributes are:</p> <ul style="list-style-type: none"> • <code>mybuffer.appendmode = 1 (ON) or 0 (OFF)</code> – default 0 over a bus interface, but 1 for ones created on the front panel • <code>mybuffer.basetimeseconds</code> returns the seconds for reading buffer entry 1 (read-only attribute). • <code>mybuffer.basetimefractional</code> returns the seconds and fractional seconds for reading buffer entry 1 (read-only attribute). • <code>mybuffer.capacity</code> for overall buffer size • <code>mybuffer.collecttimestamps = 1(ON) or 0(OFF)</code> – default 1 • <code>mybuffer.collectchannels = 1(ON) or 0(OFF)</code> – default 1 • <code>mybuffer.n</code> for number of readings stored in buffer currently • <code>mybuffer.timestampresolution</code> returns the resolution of the time stamping (read-only attribute). <p>The following buffer bits indicate buffer statuses:</p> <p><code>dmm.buffer.LIMIT1_LOW_BIT</code> or 1 <code>dmm.buffer.LIMIT1_HIGH_BIT</code> or 2 <code>dmm.buffer.LIMIT2_LOW_BIT</code> or 4 <code>dmm.buffer.LIMIT2_HIGH_BIT</code> or 8 <code>dmm.buffer.MEAS_OVERFLOW_BIT</code> or 64 <code>dmm.buffer.MEAS_CONNECT_QUESTION_BIT</code> or 128</p>

dmm.makebuffer()	
Details, continued	<p>To see readings in buffer:</p> <pre>printbuffer(x, y, mybuffer)</pre> <p>x and y: represent reading numbers desired</p> <p>To see readings, channels, and units:</p> <pre>printbuffer(x, y, mybuffer, mybuffer.channels, mybuffer.units)</pre> <p>x and y: represent reading numbers desired</p> <p>To see time stamps in buffer:</p> <pre>mybuffer.collecttimestamps = 1 print(x, y, mybuffer, mybuffer.timestamps)</pre> <p>x and y: represent readings and time stamps for elements x to y</p> <p>To see seconds, fractional seconds, and relative time stamps,</p> <pre>mybuffer.collecttimestamps = 1 printbuffer(x,y, mybuffer.seconds) printbuffer(x,y, mybuffer.fractionalseconds) printbuffer(x,y, mybuffer.relativetimestamps)</pre>
Also see	Reading buffers (on page 7-12) for more information on reading buffer aspects in the system
Example	<p>To create a user reading buffer named <code>mybuffer2</code>, with a capacity of 300:</p> <pre>mybuffer2 = dmm.makebuffer(300)</pre> <p>To delete <code>mybuffer2</code>:</p> <pre>mybuffer2 = nil</pre>
dmm.savebuffer()	
Function	Saves data from the specified dynamically-allocated buffer to the USB flash drive using the specified filename.

dmm.savebuffer()	
Usage	<p><code>dmm.savebuffer('<reading buffer name>', '<filename>', time_format)</code></p> <p>reading buffer name: The name of a previously created DMM reading buffer, specified as a string. Do not pass the reading buffer name without quotes because this generates a data type error. For example, if the reading buffer is <code>mybuffer</code>, then the buffer name should be specified as <code>"mybuffer"</code> and not <code>mybuffer</code>.</p> <p>filename: The destination filename located on the USB flash drive. The filename must specify the full path (including <code>/usb1/</code>) and include the name of file with the file extension <code>.csv</code>. If no file extension is specified, <code>.csv</code> will be added to filename.</p> <p>time_format: This optional parameter indicates what date and time information should be saved in the file to the thumb drive. Use the following values for <code>time_format</code>:</p> <ul style="list-style-type: none"> • <code>dmm.buffer.SAVE_RELATIVE_TIME</code>, which saves relative time stamps only • <code>dmm.buffer.SAVE_FORMAT_TIME</code>, which is the default if no time format specified and saves dates, times and fractional seconds • <code>dmm.buffer.SAVE_RAW_TIME</code>, which saves seconds and fractional seconds only • <code>dmm.buffer.SAVE_TIMESTAMP_TIME</code>, which only saves time stamps <p>For options that save more than one item of time information, each item is comma delimited. For example, <code>dmm.buffer.SAVE_FORMAT_TIME</code> will have <code><date></code>, <code><time></code>, and <code><fractional seconds></code> for each reading.</p>
Remarks	<p>The first parameter (reading buffer name) represents the reading buffer to be saved. The second (filename) is the filename of file to save reading buffer data to on USB flash drive. The third parameter is optional and indicates how the date and time information from the buffer should be saved. For options that save more than one item of time information, each item is comma delimited. For example, the default format will have <code><date></code>, <code><time></code>, and <code><fractional seconds></code> for each reading.</p> <p>Errors will be generated if reading buffer does not exist or is not a DMM buffer, or if the destination filename is not specified correctly. The <code>.csv</code> is appended to the filename (unless the <code>.csv</code> is specified by user). Any specified file extension other than <code>.csv</code> will generate errors.</p> <p>Valid destination filename examples:</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata')</pre> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.csv')</pre> <p>Invalid destination filename examples:</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.')</pre> <p>-Invalid extension due to period by no following letters for extension.</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.txt')</pre> <p>-Invalid extension. Use <code>.csv</code> or do not specify (no period)</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.txt.csv')</pre> <p>-invalid extension because 2 periods specified (<code>mydata_txt.csv</code> would be correct).</p>

dmm.savebuffer()	
Example	<p>To save readings from valid DMM buffer named <code>mybuffer</code> with default time information to a file named <code>mydata.csv</code> on the USB flash drive:</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.csv')</pre> <p>To save readings from <code>mybuffer</code> with relative time stamps to a file named <code>mydatarel.csv</code> on the USB flash drive:</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydatarel.csv', dmm.buffer.SAVE_RELATIVE_TIME)</pre>

Reading buffers

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

Reading buffer designations

To access the buffer, include the buffer attribute in the respective command. For example, the following commands would store five readings from the DMM into a buffer named `readingbuffer`:

```
-- Sets how many readings to take with the dmm.measure
  command.
dmm.measurecount = 5

-- Takes the measurements and stores them in readingbuffer.
dmm.measure(readingbuffer)
```

NOTE Do not use quotes around the reading buffer name when you send the `dmm.measure (readingbuffer)` command from the instrument front panel, because a data type error message will be logged to the error queue.

Buffer storage control attributes

Buffer storage attributes are summarized in the following table. To control which elements are stored in the buffer, enable the desired attribute for the buffer (which sets it to 1). The following attributes are all available per reading buffer. For example, to access the `appendmode` attribute for a buffer named `rb`, send `rb.appendmode`.

Attribute	Description
<code>appendmode</code>	<p>When off, a new measurement to this buffer will clear the previous contents before storing the new measurement. When on, the first new measurement will be stored at what was formerly <code>rb[n+1]</code>.</p> <p>This attribute is initialized to off when the buffer is created over the bus. However, the default is on for the front panel or web interface to allow triggered readings to fill a buffer without clearing the previous ones.</p>
<code>collectchannels</code>	<p>When on, channel or channel pattern information is stored with readings in the buffer. This requires eight extra bytes of storage per reading.</p> <p>This value, <code>off</code> or <code>on</code>, can only be changed when the buffer is empty (cleared). When the buffer is created, this attribute is initialized to on.</p>
<code>collecttimestamps</code>	<p>When on, time stamps will be stored with readings in the buffer. This requires eight extra bytes of storage per reading.</p> <p>This value, <code>off</code> or <code>on</code>, can only be changed when the buffer is empty (cleared). When the buffer is created, this attribute is initialized to on.</p>

Buffer read-only attributes

Use buffer read-only attributes to access the information contained in an existing buffer. The following attributes are available per reading buffer (for example, `rb.basetimeseconds` would access `basetimeseconds` for reading buffer `rb`, and the number of readings the reading buffer can store is accessed as `rb.capacity`).

Attribute	Description
<code>basetimefractional</code>	The fractional portion of the time stamp of when the reading at <code>rb[1]</code> was stored in the reading buffer (in seconds).
<code>basetimeseconds</code>	The seconds portion of the time stamp, in whole seconds, when the reading at <code>rb[1]</code> was stored in the buffer.
<code>capacity</code>	The total number of readings that can be stored in the reading buffer.
<code>timestampresolution</code>	The time stamp resolution, in seconds. The resolution is fixed at 1e-9 seconds.

Buffer programming examples

Refer to the following for buffer control programming examples. In the example, the buffer is named `readingbuffer`.

NOTE You must clear the buffer using the `readingbuffer.clear()` command before changing buffer control attributes.

Command	Description
<code>readingbuffer.collectchannels = 1</code>	Enable channel storage.
<code>readingbuffer.appendmode = 1</code>	Enable the buffer append mode.
<code>readingbuffer.collecttimestamps = 0</code>	Disable time stamp storage.

Refer to the following for buffer read-only attribute programming examples. In the example, the buffer is named `readingbuffer`.

Command	Description
<code>number = readingbuffer.n</code>	Request number of readings stored in the buffer.
<code>buffer_size = readingbuffer.capacity</code>	Request the buffer storage capacity.

Buffer reading attributes

The table in [Buffer recall attributes](#) (on page 7-14) lists the attributes that control which elements are recalled from the buffer. To access specific elements, append the desired attribute to the buffer designation.

For example, the following command line returns 100 readings from `readingbuffer1`:

```
printbuffer(1, 100, readingbuffer1.readings)
```

Similarly, the following command line returns 100 channel values from `readingbuffer1`:

```
printbuffer(1, 100, readingbuffer1.channels)
```

The default reading buffer recall attribute is `readings`, which can be omitted. Thus, the following command line also returns 100 readings from `readingbuffer1`:

```
printbuffer(1, 100, readingbuffer1)
```

Buffer recall attributes

The following table lists the attributes that control which elements are recalled from the buffer. Each is actually a nested table. Related entries are stored at the same index as the relevant measurement.

NOTE The default attribute is `readings` and can be omitted. For example, `readingbuffer1` and `readingbuffer1.readings` will both return readings from the buffer named `readingbuffer1`.

Recall attribute	Description
channels	<p>An array (a Lua table) of strings indicating the channel or channel pattern associated with the measurement.</p> <p>The returned value provides different information, based on what was opened or closed when the reading was acquired:</p> <ul style="list-style-type: none"> • If no channel or channel pattern is closed when the reading was acquired, "None" is displayed. • If only a single channel or backplane relay was closed, the channel number is displayed (for example, 5003 or 5915). • If a channel or backplane relay plus another backplane relay or other channel is closed, then the channel number will be displayed followed by a + sign (for example, 3005+ or 3915+). The channel will be in the image unless the last close operation involved only backplane relays. • If multiple channels and/or backplane relays were closed in a channel list, the last channel specified will be stored. Channels take precedence over backplane relays when stored. However, if only multiple backplane relays are specified, then the first one will be stored. • If a channel pattern was closed, then the first eight characters of the channel pattern name are returned (for example, <code>mypattern1</code> is shown as <code>mypatter</code>).
dates	An array (a Lua table) of strings, indicating the date of the reading formatted in month, day, and year.
formattedreadings	An array (a Lua table) of strings indicating the formatted reading as viewed on the display.
ptpseconds	An array (a Lua table) of the seconds portion of the time stamp of when the reading was stored. These seconds are absolute and in PTP format.
readings	An array (a Lua table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly, that is, <code>rb[2]</code> and <code>rb.readings[2]</code> are the same value.
relativetimestamps	An array (a Lua table) of time stamps, in seconds, of when each reading occurred relative to the time stamp of reading buffer entry number 1. These are equal to the time that has lapsed for each reading since the first reading was stored in the buffer. Therefore, the relative time stamp for entry number 1 in the buffer will equal 0.
statuses	An array (a Lua table) of status values for all readings in the buffer. The status values are floating-point numbers that encode the status value into a floating-point value (see the table in Buffer status (on page 7-16)).
times	An array (a Lua table) of strings, indicating the time of the reading formatted in hours, minutes, and seconds.

Recall attribute	Description
timestamps	An array (a Lua table) of strings, indicating the time stamp of the reading formatted in month, day, year, hours, minutes, seconds, and fractional seconds.
fractionalseconds	An array (a Lua table) of the fractional portion of the time stamps, in seconds, of when each reading occurred. These are absolute fractional times.
seconds	An array (a Lua table) of the seconds portion of the time stamp when the reading was stored, in seconds. These seconds are absolute and in UTC format.
units	An array (Lua table) of the strings, indicating the unit of measure stored with readings in the buffer. Units may be designated as one of the following: 'Volts AC', 'Volts DC', 'Amps AC', 'Amps DC', 'dB VAC', 'dB VDC', 'Ohms 2wire', 'Ohms 4wire', 'Ohms ComSide', 'Fahrenheit', 'Kelvin', 'Celsius', 'Hertz', 'Seconds', and 'Continuity'.

Example to access recall attributes

To see seconds, fractional seconds, and relative time stamps for entry numbers 2 and 3 in buffer `mybuffer2`, assuming `mybuffer2.collecttimestamps` was set to 1 when the readings were stored (`mybuffer2.collecttimestamps = 1`):

```
printbuffer(2,3, mybuffer2.seconds)
printbuffer(2,3, mybuffer2.fractionalseconds)
printbuffer(2,3, mybuffer2.relativetimestamps)
```

Time and date values

Time and date values are represented as a number of UTC seconds since 12:00 a.m. Jan. 1, 1970. The `os.time()` command returns values in this format. Use `os.date()` to return values in month, day, year, hours, and minutes format, or to access the time stamp table. The only exception to this is the use of the `ptpseconds` recall attribute, which has the seconds in PTP format.

Buffer status

The buffer reading status attribute can include the status information as a numeric value shown in the following table. To access status information, send the following command:

```
stat_info = readingbuffer.statuses[2]
```

Buffer status bits

Bit	Name	Hex value	ICL
B0	Low limit 1	0x01	dmm.buffer.LIMIT1_LOW_BIT
B1	High limit 1	0x02	dmm.buffer.LIMIT1_HIGH_BIT
B2	Low limit 2	0x04	dmm.buffer.LIMIT2_LOW_BIT
B3	High limit 2	0x08	dmm.buffer.LIMIT2_HIGH_BIT
B6	Measure overflow	0x40	dmm.buffer.MEAS_OVERFLOW_BIT
B7	Measure connect question	0x80	dmm.buffer.MEAS_CONNECT_QUESTION_BIT

Dynamically-allocated buffers

RAM reading buffers are created and dynamically allocated with the `dmm.makebuffer(n)` command, where **n** is the maximum number of readings the buffer can store.

Example:

To allocate a buffer named `mybuffer` that can store 100 readings:

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

Example:

To delete an allocated buffer named `mybuffer`:

```
mybuffer = nil
```

Example:

To see if the high limit 1 was exceeded during the reading:

```
stat_info = readingbuffer.statuses[3]
if (bit.band(stat_info, dmm.buffer.LIMIT1_HIGH_BIT) ==
    dmm.buffer.LIMIT1_HIGH_BIT) then
    print("Limit 1 high exceeded")
else
    print("Limit 1 high okay")
end
```

Dynamic buffer programming example

The programming example below shows how to store data using a dynamically-allocated buffer named `mybuff`.

```
-- Reset the DMM.
dmm.reset('all')

-- Create a buffer named mybuffer and allocate space for
  100,000 readings.
mybuffer = dmm.makebuffer(100000)

-- Enable append buffer mode.
mybuffer.appendmode = 1

-- Set count to 1.
dmm.measurecount = 1

-- Select the DMM function as DC volts.
dmm.func = dmm.DC_VOLTS

-- Start for...do loop. Measure and store readings in buffer.
  End loop.
for x = 1, 100 do
  dmm.measure(mybuffer)
end

-- Return readings 1-100.
printbuffer(1, 100, mybuffer.readings)

-- Return units 1-100.
printbuffer(1, 100, mybuffer.units)
```

Buffer for...do loops

The following examples illustrate the use of for...do loops with respect to recalling buffer data from a reading buffer called `mybuffer`. The following code may be sent as one command line or as part of a script. Sample outputs follow the line of code. Also see the `printbuffer()` (on page 13-221) ICL command.

NOTE Buffer `mybuffer` has time stamp collection enabled in the example below.

This example loop uses `printbuffer` to show the reading, units, and relative time stamps for all readings stored in the buffer. The information for each reading (reading, units, and relative time stamps) is shown on a single line with the elements comma-delimited.

```
for x = 1,mybuffer.n do
  printbuffer(x,x,mybuffer, mybuffer.units,
             mybuffer.relativetimestamps)
end
```

Sample comma-delimited output of above code:

```
3.535493836e-002, Volts DC, 0.000000000e+000
-4.749810696e-002, Volts DC, 5.730966000e-002
-8.893087506e-002, Volts DC, 7.722769500e-002
4.164193198e-002, Volts DC, 1.246876800e-001
-6.900507957e-002, Volts DC, 1.815213600e-001
-8.851423860e-002, Volts DC, 2.009161500e-001
3.891038895e-002, Volts DC, 2.647790700e-001
-7.581630349e-002, Volts DC, 3.032140350e-001
-8.236359060e-002, Volts DC, 3.226125750e-001
-8.551311493e-002, Volts DC, 3.425625900e-001
```

The following loop uses the `print` command instead of the `printbuffer` command. This loop shows the same information described in the previous example (reading, units, and relative time stamps for all readings stored in the buffer). However, because the `print` command is used over `printbuffer`, each line is tab-delimited (rather than comma-delimited) to produce a columnar output, as shown below:

```
for x = 1,mybuffer.n do
  print(mybuffer.readings[x], mybuffer.units[x],
        mybuffer.relativetimestamps[x])
end
```

Sample columnar output of above code:

3.535493836e-002	Volts DC	0.000000000e+000
-4.749810696e-002	Volts DC	5.730966000e-002
-8.893087506e-002	Volts DC	7.722769500e-002
4.164193198e-002	Volts DC	1.246876800e-001
-6.900507957e-002	Volts DC	1.815213600e-001
-8.851423860e-002	Volts DC	2.009161500e-001
3.891038895e-002	Volts DC	2.647790700e-001
-7.581630349e-002	Volts DC	3.032140350e-001
-8.236359060e-002	Volts DC	3.226125750e-001
-8.551311493e-002	Volts DC	3.425625900e-001

If data was collected by executing a three-channel scan list with a scan count of 10, the buffer has 30 readings in it. To see the comma-delimited data on the three-channel boundary:

```
x = 1
y = 3
for z = 1, 10 do
  printbuffer(x, y, mybuffer, mybuffer.channels)
  x = x + 3
  y = y + 3
end
```


The sample output from the above code has six comma-delimited entries per line (reading, channel, reading, channel, reading, channel):

```
3.181298825e-002, 2001+, -5.602844334e-002, 2002+, -7.811298360e-002,
2003+
3.228547367e-002, 2001+, -5.299202901e-002, 2002+, -8.676257870e-002,
2003+
3.736769697e-002, 2001+, -3.247188344e-002, 2002+, -5.106155438e-002,
2003+
-6.473406636e-002, 2001+, -9.218081926e-002, 2002+, 3.419026595e-002,
2003+
-3.856921662e-002, 2001+, -6.672781529e-002, 2002+, -7.762540017e-002,
2003+
2.876431571e-002, 2001+, -4.056434134e-002, 2002+, -6.119288115e-002,
2003+
-7.301064720e-002, 2001+, 2.893913659e-002, 2002+, -3.164065858e-002,
2003+
-6.794576932e-002, 2001+, -8.067066262e-002, 2002+, 2.339088329e-002,
2003+
-5.288247880e-002, 2001+, -6.769966949e-002, 2002+, -7.572277347e-002,
2003+
2.618149827e-002, 2001+, -3.164126270e-002, 2002+, -6.306067024e-002,
2003+
```

If you want to see more information about the readings, add the appropriate buffer recall attribute to the `printbuffer` line in the sample code. For example, to see the relative time stamp with each reading, add `mybuffer.relativetimestamps` to the `printbuffer` command as follows:

```
printbuffer(x, y, mybuffer, mybuffer.channels,
            mybuffer.relativetimestamps)
```

In the output from this `printbuffer` command, nine comma-delimited entries appear on each line. Each line will include the following entries: reading, channel, relative time stamp, reading, channel, relative time stamp, reading, channel, relative time stamp.

Exceeding reading buffer capacity

If the reading buffer count is not exceeded, readings are stored as expected. But if the reading buffer capacity would be exceeded by new readings being added to the current buffer index, the count is lowered to a new count so it does not exceed the buffer capacity. Once the buffer is full (to the new count), no more readings are taken and error message 4915 is displayed, stating that you attempted to exceed the capacity of the reading buffer. If you attempt to store additional readings in a full buffer, the same message appears, and no readings are taken.

Example:

Create a buffer with:

- A capacity for 50 readings
- Append mode enabled
- Measure count to 30

Tell the instrument to print the current number of buffer elements stored and take readings to store in the buffer. The following occurs:

1. The first time the measurement is called, the buffer is empty (no readings) so it stores 30 readings.
2. The second time the measurement is called it stores only 20 readings. This is because $30 + 30$ is 60 readings, which exceeds buffer capacity (50). Because 30 readings are already stored, only 20 readings are taken and stored. Error message 4915 is displayed.
3. The third time the measurement is called, the buffer is full (already has 50 readings). Because there is no more room, no readings are taken (nil response for reading) and error message 4915 is again displayed.

The following listing provides the coding for the previous example:

```
-- Create a buffer named buf and allocate space for 50
readings.
buf = dmm.makebuffer(50)

-- Enable append buffer mode.
buf.appendmode = 1

-- Set count to 30.
dmm.measurecount = 30

-- Show the current number of readings in the buffer, and
then measure and store readings in the buffer (first
pass).

-- Output from the print command:
-- 0.000000000e+000
-- 5.245720223e-002
print(buf.n, dmm.measure(buf))

-- Show the current number of readings in the buffer, and
then measure and store readings in the buffer (second
pass).

-- Output from the print command:
-- 3.000000000e+001
-- -1.388141960e-001
-- 4915, Attempting to store past capacity of reading
buffer
print(buf.n, dmm.measure(buf))
```

```
-- Show the current number of readings in the buffer, and
  then measure and store readings in the buffer (third
  pass).
-- Output from the print command:
-- 5.000000000e+001
-- nil
-- 4915, Attempting to store past capacity of reading
  buffer
print(buf.n, dmm.measure(buf))
```


In this section:

Scanning fundamentals.....	8-1
Scan and step counts.....	8-7
Basic scan procedure.....	8-7
Front panel scanning.....	8-10
Bus operation scanning.....	8-12
Hardware trigger modes.....	8-18

Scanning fundamentals

A scan is a series of steps which open and close switches to be optionally measured. The step-by-step flow is defined by the trigger model.

The Keithley Instruments Series 3700 System Switch/Multimeter can scan channels with up to six Keithley Instruments switching modules installed. Each scan channel can have its own unique setup. Aspects of operation that may be uniquely set for each channel include function, range, rate, AC bandwidth, REL, filter, digits, math, offset compensation, temperature transducers, limits, volts dB, and so on.

NOTE If desired, readings for scanned channels may be automatically stored in a specified reading buffer (see [Buffer: Data Storage and Retrieval](#) (on page 7-1)).

You can configure and execute scans from the front panel, remotely over the bus, or through the Series 3700 web interface. The steps are executed in the order that they are added. When adding a range of channels, they are added to the end of the existing scan list.

For example:

- `1003:1005` will add Channels 1003, 1004, and 1005 to the list as three distinct steps, with Channel 3 added first, Channel 4 added second, and Channel 5 added third.
- Adding individual channels in the order of 1003, 1005, and 1004 will add the channels to the list as three distinct steps with Channel 3 added first, Channel 5 added second, and Channel 4 added last.

Channel assignments

Each switching module has a certain number of channels. For example, the Model 3720 switching module has 60 channels (1 through 60). When you encounter a 1- to 3-digit channel number in this manual, the switching module channel is the point of discussion. A 4-digit channel number includes the slot number followed by the 3-digit channel number.

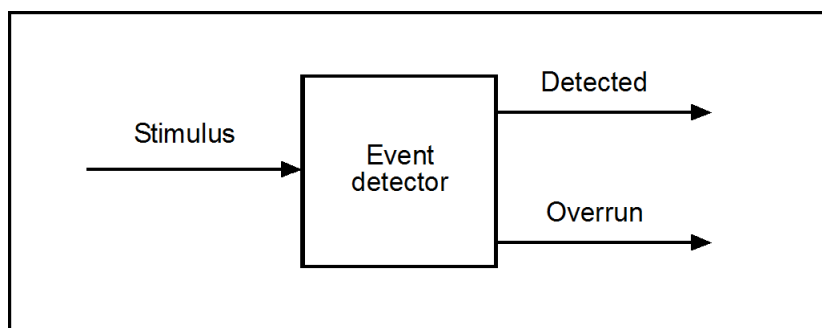
A switching module can be installed in any of the mainframe's six slots. Therefore, to close, open, or scan a channel, you must specify the slot location and channel number of the switching module by using a four-digit channel number for the mainframe. The first digit (1, 2, 3, 4, 5, or 6) indicates the slot number, and the next three digits indicate one of the following:

- The MUX (multiplexer) channel notation
- The row and column of matrix card notation
- The modules' backplane relay notation

Events

Event detectors monitor an event. They have one input signal (the stimulus), which is the event that they monitor (in some cases, the stimulus is an action in the system, like a timer expiring or a key press). They have two optional output signals (see figure below). "Detected" reflects the detection state of the event detector. If an event was detected, the detected signal is asserted. Event detectors are usually coupled to something that consumes the events. When an event is consumed, the detected state of the event detector is reset. Should an event be detected while the event detector is in the detected state, the overrun signal will be asserted. You can only clear the overrun signal by sending an ICL command.

Figure 8-1: Event detector



Event blenders

Advanced event handling requires a way to wait for one of several events (or all of several events). An event blender provides for this combining or blending of events. An event blender can combine up to four events in either an "or" mode or an "and" mode. When in "or" mode, any one of the input events will cause an output event to be generated. When in "and" mode, all the input events must occur before an output event is generated.

When operating in "and" mode, if an event is detected more than once before all events necessary for the generation of an output event, an action overrun will be generated. When operating in "or" mode, an action overrun will be generated when two or more source events are detected simultaneously.

Event blenders each have an associated event detector that can be accessed through script control. Event blenders can only be accessed over the bus (no front panel control is available). The following ICL commands provide additional information on available blenders:

- [*trigger.blender\[N\].clear\(\)*](#) (on page 13-287)
- [*trigger.blender\[N\].orenable*](#) (on page 13-287)
- [*trigger.blender\[N\].overrun*](#) (on page 13-288)
- [*trigger.blender\[N\].stimulus\[M\]*](#) (on page 13-288)
- [*trigger.blender\[N\].wait\(\)*](#) (on page 13-289)

Foreground and background scan execution

You can execute a scan in the foreground or background. Background execution allows you to query settings or access reading buffer data. If a scan is running in the foreground, it will need to finish or be aborted before you can query any settings or access reading buffers.

When a scan is running in the background, you can send ICL commands to be processed. The commands that you can use include most of the command messages that you use to query for settings, for example:

```
print(dmm.func)
print(dmm.scan.state())
printbuffer(1, 5, rb)
print(scan.state())
```

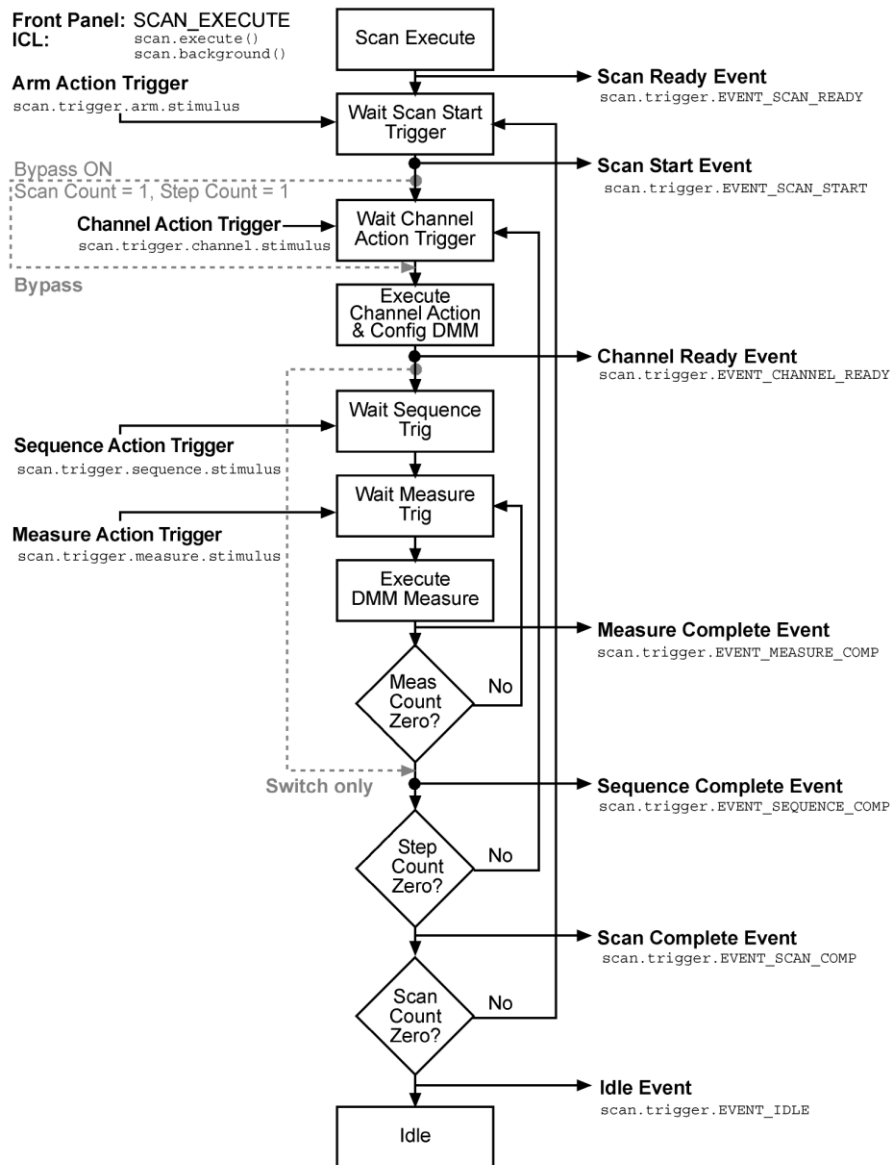
Most of the commands to change how the instrument is configured will log the following error message to the error queue, if not allowed:

```
5522, Scan Running, Must Abort Scan
```

Trigger model

The following flowchart represents a trigger model implemented in the Series 3700. The trigger model is used during a scan only. For front panel operation, you use the **SCAN** and **STEP** keys to perform scan actions. For remote operation, you use the scan functions and attributes commands, for example, `scan.execute()` and `scan.mode`.

Figure 8-2: Trigger model



The Series 3700 trigger model has the following events and associated ICL command attributes. These events, along with other events in the system, may be used to configure various stimulus settings.

For example, the channel ready event (`scan.trigger.EVENT_CHANNEL_READY`) may be set to pulse digital I/O line 3 when it gets generated. The command message for this would be:

```
digio.trigger[3].stimulus =
    scan.trigger.EVENT_CHANNEL_READY
```

Likewise, you can use the digital I/O line 5 trigger event to satisfy the scan trigger channel stimulus, which causes the channel action to occur when a trigger is detected on line 5. The command message for this is:

```
scan.trigger.channel.stimulus = digio.trigger[5].EVENT_ID
```

Event	Associated ICL attribute
Scan Ready Event	<code>scan.trigger.EVENT_SCAN_READY</code>
Scan Start Event	<code>scan.trigger.EVENT_SCAN_START</code>
Channel Ready Event	<code>scan.trigger.EVENT_CHANNEL_READY</code>
Measure Complete Event	<code>scan.trigger.EVENT_MEASURE_COMP</code>
Sequence Complete Event	<code>scan.trigger.EVENT_SEQUENCE_COMP</code>
Scan Complete Event	<code>scan.trigger.EVENT_SCAN_COMP</code>
Idle Event	<code>scan.trigger.EVENT_IDLE</code>

NOTE Scanning operations run through the trigger model, but individual open, close, and DMM measure commands have no interaction with the trigger model.

Trigger model components

The individual components of the trigger model are explained in the following paragraphs.

Idle

When a scan is initiated, the operation leaves the idle state and finalizes aspects to start scanning. Once everything is ready for scanning, the instrument generates the Scan Ready Event and waits for the arm stimulus event (see the Arm Action Trigger in the trigger model figure in [Trigger model](#) (on page 8-4)). After the last channel in the scan is measured, operation returns to the idle state, where measurements are halted and the last channel in the list is closed.

Triggers

The following four action triggers are associated with the trigger model operation:

- Arm Action Trigger
- Channel Action Trigger (this trigger may be bypassed for the first channel of the first scan count)
- Sequence Action Trigger
- Measure Action Trigger

STEP operation: When the trigger event is detected, a channel is closed and a measurement may be taken, if it is configured for one. The step operation, which is only available over the front panel, processes one step at a time until all steps are completed.

SCAN operation: When the trigger event is detected, all the channels in the scan list are scanned. The scan operation processes all steps before finishing.

The channel closing occurs when the channel stimulus event is detected. After closing the channels, the Channel Ready Event is generated. See the Channel Action Trigger in the trigger model figure in [Trigger model](#) (on page 8-4). Before the measurement occurs, the sequence and measure stimulus events must be detected. One or both of these events may be used.

After the measurement is completed, a Measure Event Complete message is generated. If a sequence of measurements is taken, a Sequence Complete Event message is generated after all measurements are taken (that is, the measure count reaches zero). Use the sequence stimulus event if you want each measurement to be paced by an event. Otherwise, use the measure event stimulus to have a single event trigger all of the measure count readings.

When all steps are complete, the Scan Complete Event message is generated. Otherwise, it loops back to the Channel Action Trigger. When all scans are complete, the Idle Event is generated. Otherwise, it loops back to the Arm Action Trigger.

Stimulus	Description
Arm Action Trigger	Affects the "Wait Scan Start Trigger" block of the trigger model. This trigger is associated with the ICL <code>scan.trigger.arm.stimulus</code> (on page 13-244).
Channel Action Trigger	Affects the "Wait Channel Start Trigger" block of the trigger model. This trigger is associated with the ICL <code>scan.trigger.channel.stimulus</code> (on page 13-246).
Sequence Action Trigger	Affects the "Wait Sequence Trigger" block of the trigger model. This trigger is associated with the ICL <code>scan.trigger.sequence.stimulus</code> (on page 13-249).

Stimulus	Description
Measure Action Trigger	Affects the "Wait Measure Trigger" block of the trigger model. This trigger is associated with the ICL <code>scan.trigger.measure.stimulus</code> (on page 13-247).

Scan and step counts

When running a scan, it may be necessary to determine the scan progress. Use scan and step count to determine the point in the scan table being executed.

"Scan count" represents the number of the current iteration through the scan portion of the trigger model. This number does not increment until after the scan begins. Therefore, if a unit is waiting for an input to trigger a scan start, the scan count will represent the previous number of scan iterations. If no scan has yet to begin, the scan count will be zero.

"Step count" represents the number of times the scan has completed a pass through the channel action portion of the trigger model. This number does not increment until after the action completes. Therefore, if the unit is waiting for an input to trigger a channel action, the step count will represent the previous step. If no step has yet to complete, the step count will be zero. If the step count has yet to complete the first step in a subsequent pass through a scan, the scan count represents the last step in the previous scan pass.

Basic scan procedure

To perform a scan:

1. Configure the channels for scanning as needed. Select (or create, if necessary) the reading buffer to store measurements (if desired).
2. Build the scan list:
 - **Front panel:** Press the **INSERT** key.
 - **Bus:** Send the `scan.create()` or `scan.add()` command.
3. Configure the scan settings (for example, scan count, bypass, mode, and so on).
4. To start the scan:
 - **Front panel:** Press the **STEP** key or the **SCAN** key and select the **BACKGROUND** menu item.
 - **Bus:** Send an ICL command such as `scan.execute`, `scan.background`, `scan.nobufferbackground`, or other appropriate scan command.

5. The trigger model leaves the idle state and opens channels involved in scanning, along with channels that would interfere with scanning, such as AMP channels, analog backplane relays 1 and 2 on all slots, common-side ohm backplane channels, and other channels in banks involved in scanning.

When you press the **STEP** key, the Series 3700 leaves the idle state and performs the channel action associated with the first step in the scan list.

Measurements are then taken (if part of the scan). If a reading buffer was selected, the result from the measurements are stored there. The measurement action, if started, is completed. The channel and DMM remain as previously configured until the next step in the scan is initiated. The DMM configuration changes to the attribute settings tied to the channel in the next step.

NOTE While scanning is enabled, pressing most front panel keys will display the message "ERROR CODE: 5522 Scan Running, Must Abort Scan."

6. The channels are scanned or stepped in the order they were added to the list.
 - **Front panel:** If you are stepping through the scan, press the **STEP** key to proceed to the next step in the list.
 - **Bus:** You cannot step a scan remotely over the bus.
7. To abort the scan:
 - **Front panel:** Press the **EXIT** key.
 - **Bus:** Use the `scan.abort` ICL command.

NOTE Even if the scan is aborted, the DMM remains as configured in the last completed step of a scan that involved measuring. Channel states match the aborted state of channels in terms of which are closed and opened.

The DMM remains as previously configured in the last completed measurement step of a scan that involved measuring. The function associated with that configuration will have the associated DMM attributes updated to match. All other functions will remain as configured prior to scanning.

If programmed to scan the channels in the scan list again, the Series 3700 will wait at the control source for another trigger event. After all the scan list channels are measured again, the Series 3700 will output another trigger pulse, if configured to do so. After all programmed scans are completed, the instrument returns to the idle state with the channels associated with last scan step closed.

Buffer

To recall scanned readings stored in the buffer, press the **REC** key and turn the navigation wheel to navigate through the buffer. See [Recalling readings](#) (on page 7-5) for details on recalling buffer readings. When finished, make sure to exit from buffer recall (press the **EXIT** key). Also see [Buffer: Data Storage and Retrieval](#) (on page 7-1).

Changing channel and DMM attributes of an existing scan

When a scan already exists, changing channel and DMM attributes also causes the scan to change. Once a scan list has been defined, the Series 3700 will try to incorporate your changes into the scan. For example, changing a DMM configuration assigned to a channel used in scanning affects the scan list. But changing a DMM configuration on a channel not involved in scanning does not affect the scan list. If the change impacts the ability of the scan to function properly (such as deleting something referenced by the scan), an error message is logged.

To see how the scan list may have changed, view the current scan list. On the front panel, press the **SCAN** key and select the **LIST** option, using the navigation wheel to scroll through the options. For bus operation, use the `scan.list()` function. For performance reasons, it is always better to configure all channel and DMM attributes before creating a scan. Afterward, changes may cause the scan to take more time to modify the scan list. You can clear an existing scan list before making any changes after making a scan list. From the front panel, press the **SCAN** key and select the **CLEAR** option. For bus operation, use the `scan.create()` function.

Some changes may cause channels to be dropped from the list when they become paired with another channel for a 4-wire operation. These channels will not be added back into the list during subsequent changes that free the paired channel from a 4-wire operation. To get a recently unpaired channel back in the list, create a new scan list or add it back into the list.

For example, a scan list is comprised of Channels 1 to 60 on a Model 3720 card with the channels configured to measure DC volts. Changing Channels 1 to 30 to be configured for 4-wire ohms measurements causes the scan list to change. The scan list changes to contain Channels 1 to 30 measuring 4-wire ohms, and Channels 31 to 60 are removed because they are paired with Channels 1 to 30. If you then change Channels 1 to 60 to be configured for measuring DC volts, the scan list will still only contain Channels 1 to 30, but it will be measuring DC volts. Channels 31 to 60 are not automatically added back into the list.

The ICL commands to simulate this example follow. Assume the Model 3720 is in Slot 3:

```
-- Configure Channels 1 to 60 to measure DC volts.
dmm.setconfig('slot3', 'dcvolts')

-- Create a scan list, channels measuring DC volts.
scan.create('slot3')

-- View the scan list, 60 channels measuring DC volts.
print(scan.list())

-- Change Channels 1 to 30 to 4-wire ohms.
dmm.setconfig('slot3', 'fourwireohms')

-- List now has Channels 1 to 30 measuring 4-wire ohms.
print(scan.list())

-- Change back to DC volts on Channels 1 to 60.
dmm.setconfig('slot3', 'dcvolts')

-- List still has Channels 1 to 30, but measures DC volts.
print(scan.list())
```

Front panel scanning

After channels have been added to the scan list, press the **SCAN** key to display the SCAN ACTION MENU. If no scan list exists, pressing the **SCAN** key will briefly display "No Scan List. Use INSERT to add selection."

The menu contains the following items:

- **BACKGROUND:** Runs scan list in the background
- **CREATE:** Displays **Use <INSERT> key** (reserved for future enhancements)
- **LIST:** Displays the current scan list steps. Turn the navigation wheel to scroll through the list.
- **CLEAR:** Clears the existing scan list.
- **RESET:** Resets the unit's scan aspects, which include scan count, clearing the scan list, and scan stimulus settings like scan trigger arm.

Press the **INSERT** key to add the selected channels or pattern to the existing scan list.

Press the **DELETE** key to remove the selected channels or pattern from the existing scan list. Only the first occurrence of the selected item will be removed. For example, if Channel 3003 appears in the list three times and Channel 3003 is selected when the **DELETE** key is pushed, the first step using Channel 3003 will be removed (the remaining two will stay in the list).

When removing channels, channel patterns will not be checked to determine if the channel being removed is associated with its image. To remove a channel pattern in a scan list, select the channel pattern to be removed, and then press the **DELETE** key. Continuing the previous example of Channel 3003, if 'mypat1' is comprised of Channels '3003, 3033, 3911, and 3922' when the remove request for Channel 3003 is made, it will not remove 'mypat1' from the list. To remove 'mypat1' from list, select the channel pattern 'mypat1' and press the **DELETE** key, which removes the step and all associated channels.

Press the **STEP** key to single step through a scan list.

Scan configuration

To configure a scan from the **SCAN ATTR MENU**, while in an active scan list:

1. Press the **CONFIG** key.
2. Press the **SCAN** key. Modify any of the following menu items as desired:
 - **ADD**: Displays Use <INSERT> key. The related ICL is `scan.add`, without the optional DMM configuration.
 - **BYPASS**: Enables (ON) or disables (OFF) bypassing the first step of the first scan pass. Related ICL command: `scan.bypass` (on page 13-233).
 - **MODE**: Sets the scan mode value to one of the following:
 - OPEN_ALL (default setting)
 - OPEN_SELECT
 - FIXED_ABRRelated ICL command: `scan.mode()` (on page 13-239).
 - **MEAS_CNT**: Sets the measure count value. Related ICL command: `scan.measurecount` (on page 13-238).
 - **SCAN_CNT**: Sets the scan count value. Related ICL command: `scan.scancount` (on page 13-242).
3. Press the **EXIT** key to leave the menu.

Bus operation scanning

ICL commands

The following list contains ICL commands associated with triggers and bus operation scanning:

- [*trigger.blender\[N\].clear\(\)*](#) (on page 13-287)
- [*trigger.blender\[N\].orenable*](#) (on page 13-287)
- [*trigger.blender\[N\].overrun*](#) (on page 13-288)
- [*trigger.blender\[N\].stimulus\[M\]*](#) (on page 13-288)
- [*trigger.blender\[N\].wait\(\)*](#) (on page 13-289)
- [*trigger.timer\[N\].clear*](#) (on page 13-290)
- [*trigger.timer\[N\].stimulus*](#) (on page 13-292)
- [*digio.trigger\[N\].clear\(\)*](#) (on page 13-88)
- [*digio.trigger\[N\].pulsewidth*](#) (on page 13-90)
- [*digio.trigger\[N\].stimulus*](#) (on page 13-91)
- [*digio.trigger\[N\].wait*](#) (on page 13-92)
- [*lan.trigger\[N\].assert\(\)*](#) (on page 13-203)
- [*lan.trigger\[N\].clear*](#) (on page 13-204)
- [*lan.trigger\[N\].overrun*](#) (on page 13-206)
- [*lan.trigger\[N\].stimulus*](#) (on page 13-208)
- [*lan.trigger\[N\].wait*](#) (on page 13-209)
- [*scan.add\(\)*](#) (on page 13-230)
- [*scan.background\(\)*](#) (on page 13-232)
- [*scan.bypass*](#) (on page 13-233)
- [*scan.create\(\)*](#) (on page 13-234)
- [*scan.execute\(\)*](#) (on page 13-236)
- [*scan.list\(\)*](#) (on page 13-237)

- [*scan.measurecount*](#) (on page 13-238)
- [*scan.mode\(\)*](#) (on page 13-239)
- [*scan.nobufferbackground\(\)*](#) (on page 13-240)
- [*scan.reset\(\)*](#) (on page 13-242)
- [*scan.scancount*](#) (on page 13-242)
- [*scan.state\(\)*](#) (on page 13-243)
- [*scan.trigger.arm.clear\(\)*](#) (on page 13-244)
- [*scan.trigger.arm.set\(\)*](#) (on page 13-244)
- [*scan.trigger.arm.stimulus*](#) (on page 13-244)
- [*scan.trigger.channel.clear\(\)*](#) (on page 13-245)
- [*scan.trigger.channel.set\(\)*](#) (on page 13-245)
- [*scan.trigger.channel.stimulus*](#) (on page 13-246)
- [*scan.trigger.clear\(\)*](#) (on page 13-247)
- [*scan.trigger.measure.clear\(\)*](#) (on page 13-247)
- [*scan.trigger.measure.set\(\)*](#) (on page 13-247)
- [*scan.trigger.measure.stimulus*](#) (on page 13-247)
- [*scan.trigger.sequence.clear\(\)*](#) (on page 13-248)
- [*scan.trigger.sequence.set\(\)*](#) (on page 13-248)
- [*scan.trigger.sequence.stimulus*](#) (on page 13-249)

Scanning examples

The following examples assume a Keithley Instruments Model 3720 module is installed in Slot 3 of a Series 3700.

NOTE In the examples, to clear a trigger stimulus after setting, set the stimulus to 0, which returns the stimulus setting back to its factory default value, which may or may not be 0.

Example 1:

Command list to scan the entire card in a switch-only application (no measuring) that has digital I/O line 1 initiate a background scan (see the comments for other specifics).

```
-- Reset the Series 3700 to factory defaults
reset()

-- Create scan for all channels on the card installed in
  Slot 3.
scan.create('slot3')

-- Setup digital I/O line 1 to detect a falling-edge
  trigger.
digio.trigger[1].mode = digio.TRIG_FALLING

-- Use a digital I/O event as the ARM layer's stimulus.
scan.trigger.arm.stimulus = digio.trigger[1].EVENT_ID

-- Initiate the scan to execute in the background.
scan.background()
```

Example 2:

Command list to scan the entire card while measuring DC volts on each channel and storing readings in a buffer called `mybuffer` (see the comments for other specifics).

```
-- Reset the Series 3700 to factory defaults.
reset()

-- Set the range of DC volts to the 10 volt range.
dmm.range = 10

-- Set NPLC to 0.1 NPLC.
dmm.nplc = .1

-- Save the DMM configuration as "mydcv."
dmm.configure.set('mydcv')

-- Make buffer named "mybuffer" and configure it to store
  up to 1000 readings.
mybuffer = dmm.makebuffer(1000)
```

```
-- Set up digital I/O line 1 to detect a falling edge
  trigger.
digio.trigger[1].mode = digio.TRIG_FALLING

-- Set each channel so it closes with a digio 1 event
  trigger.
scan.trigger.channel.stimulus = digio.trigger[1].EVENT_ID

-- Set bypass to off so that first channel needs to see
  trigger before closing.
scan.bypass = scan.OFF

-- Create scan for Channels 1 to 60 on the card installed
  in Slot 3.
scan.create('3001:3060', 'mydcv')

-- Initiate the scan to execute in the background and save
  readings to a buffer called "mybuffer."
scan.background(mybuffer)
```

Example 3:

Command list to scan the entire card while measuring 4-wire ohms using a background scan (see the comments for other specifics).

```
-- Reset the Series 3700 to factory defaults.
reset()

-- Set the configuration for all channels in Slot 4 to 4-
  wire ohms.
dmm.setconfig('slot4', 'fourwireohms')

-- Create scan for all channels on the card installed in
  Slot 4.
scan.create('slot4')

-- Set up digital I/O Line 1 to detect a falling-edge
  trigger.
digio.trigger[1].mode = digio.TRIG_FALLING

-- Set up digital I/O Line 2 to detect a falling-edge
  trigger.
digio.trigger[2].mode = digio.TRIG_FALLING

-- Set each channel so that it will close with a
  measurement complete event.
scan.trigger.channel.stimulus =
  scan.trigger.EVENT_MEASURE_COMP

-- Set digio 2 to pulse when a channel ready event occurs.
digio.trigger[2].stimulus =
  scan.trigger.EVENT_CHANNEL_READY

-- Set each measurement to occur with a digio 1 event
  trigger.
scan.trigger.measure.stimulus = digio.trigger[1].EVENT_ID
```

```
-- Set bypass to ON so first channel closes without taking
  a measurement.
scan.bypass = scan.ON

-- Make buffer named "mybuffer" and configure it to store
  up to 1000 readings.
mybuffer = dmm.makebuffer(1000)

-- Initiate the scan to execute in the background and save
  readings to a buffer called "mybuffer."
scan.background(mybuffer)
```

Example 4:

Optimizing scanning for speed.

Some cards, such as the Model 3723, use relays that are optimized for switching speed and reliability. However, these cards still use backplane relays (EMR) that are slow and have a shorter life. Full speed and reliability of the card can be realized by avoiding scan modes that intelligently open and close backplane relays (for example, `scan.MODE_OPEN_SELECTIVE`). By setting the scan mode (`scan.mode`) to `scan.MODE_FIXED_ABR`, all required backplane relays will be closed prior to the start of the scan and remain closed until you program them to open.

Here's an example of a Model 3706 configured for fast scanning with the Model 3723 card. Sixty channels will be scanned ten times on 200V DCV.

```
-- Reset the Series 3700 to factory defaults.
reset()

-- Select active function as DC volts.
dmm.func= 'dcvolts'

-- Turn auto range off.
dmm.autorange=dmm.OFF

-- Select the range based on 200 volts.
dmm.range='200'

-- Turn autozero off.
dmm.autozero=dmm.OFF

-- Set the NPLC to .006.
dmm.nplc=.006

-- Turn auto delay off.
dmm.autodelay=dmm.OFF

-- Create a reading buffer to hold 600 readings.
reading_buffer=dmm.makebuffer(600)

-- Save the current dmm settings as "mydcvolts"
  configuration.
dmm.configure.set('mydcvolts')
```

```

-- Assign that configuration to Channels 1 to 60 on Slot 1.
dmm.setconfig( '1001:1060' , 'mydcvolts')

-- Set the scan mode to fixed ABR.
scan.mode=scan.MODE_FIXED_ABR

-- Create a scan list of Channels 1 to 60 on Slot 1.
scan.create('1001:1060')

-- Set the scan count to 10.
scan.scancount=10

-- Scan in the foreground.
scan.execute(reading_buffer)

-- Write the data out to a file on a USB flash drive.
dmm.savebuffer('reading_buffer', '/usb1/mydata.csv')

```

NOTE The NPLC setting is at .006 in the example, but the fastest NPLC setting supported in a Series 3700 is .0005. Another speed improvement option is to set the channel connect rule to OFF (`channel.connectrule = channel.OFF`). Using this setting allows channels to open and close at the same time, provided the application supports this operation.

Example 5:

Command list to scan the entire Model 3723 card while measuring DC volts on each channel, and store readings in a buffer called `mybuffer` (see the comments for other specifics).

NOTE For the Model 3723, the channels are reed relays, while the analog backplane relays are EMR relays. Therefore, to have the scan run faster, set the scan mode to fixed ABR, which closes the backplane relays before scanning starts and keeps them closed during the entire scan.

```

-- Reset the Series 3700 to factory defaults.
reset()

-- Set the range of DC volts to the 10 volt range.
dmm.range = 10

-- Set NPLC to 0.1 NPLC.
dmm.nplc = .1

-- Save the DMM configuration as "mydcv."
dmm.configure.set('mydcv')

-- Make buffer named "mybuffer" and configure it to store
  up to 1000 readings.
mybuffer = dmm.makebuffer(1000)

```

```

-- Set up digital I/O Line 1 to detect a falling edge
  trigger.
digio.trigger[1].mode = digio.TRIG_FALLING

-- Set each channel so it closes with a digio 1 event
  trigger.
scan.trigger.channel.stimulus = digio.trigger[1].EVENT_ID

-- Set bypass to OFF so that the first channel needs to see
  the trigger before closing.
scan.bypass = scan.OFF

-- Set the mode to fixed ABR so that the backplane relays
  are closed at the start of scanning and maintained
  closed throughout scanning without being opened/closed.
scan.mode = scan.MODE_FIXED_ABR

-- Create scan for Channels 1 to 60 on the card installed
  in Slot 3.
scan.create('3001:3060', 'mydcv')

-- Initiate the scan to execute in the background and save
  readings to a buffer called "mybuffer."
scan.background(mybuffer)
    
```

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 3700 contains 14 digital I/O lines and three TSP-Link synchronization lines that you can use for input or output triggering. The following table provides a summary for each hardware trigger mode.

Trigger mode	Output		Input	Notes
	Unasserted	Asserted		
	Unasserted	Asserted	Detects	
Bypass	N/A	N/A	N/A	Use the <code>writetbit</code> and <code>writeport</code> 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	<ul style="list-style-type: none"> The programmed state of the line determines if the behavior is similar to <code>RisingA</code> or <code>RisingM</code> High similar to <code>RisingA</code> Low similar to <code>RisingM</code>

Trigger mode	Output		Input	Notes
	Unasserted	Asserted		
	Unasserted	Asserted	Detects	
Rising A	High	Low	Rising	
RisingM	Low	High	None	
Synchronous	High latching	Low	Falling	<ul style="list-style-type: none"> 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.

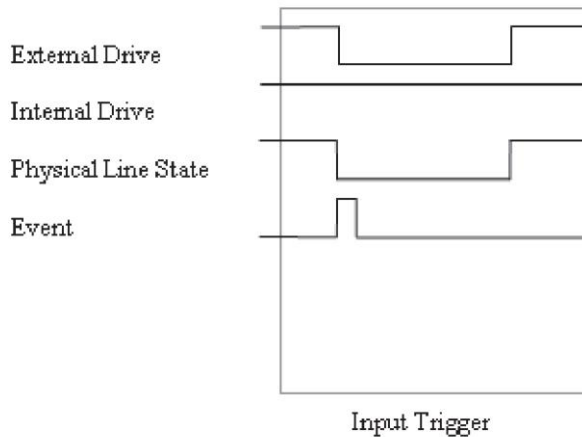
For additional information on the hardware trigger modes, see [Instrument Control Library \(ICL\)](#) (on page 12-1).

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. The following graphic illustrates the characteristics for the falling edge input trigger.

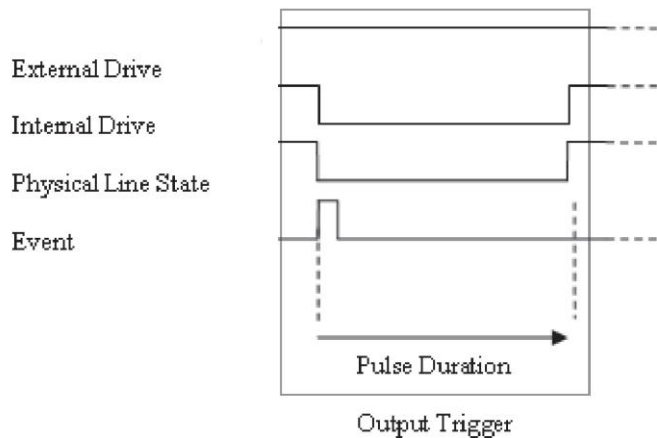
Figure 8-3: Falling edge input trigger



Input characteristics:

Detects all falling edges as input triggers

Figure 8-4: Falling edge output trigger



Output characteristics:

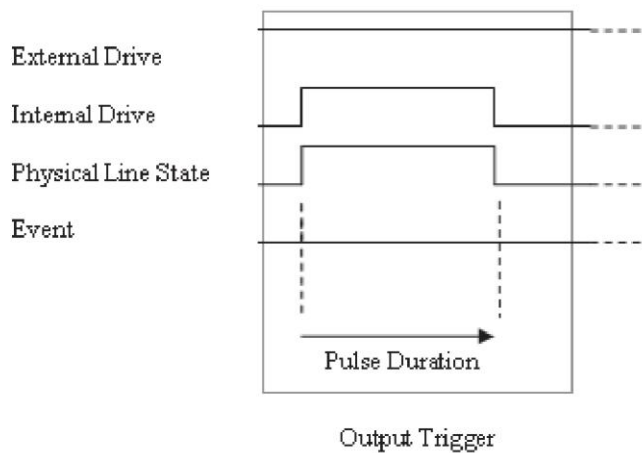
The `trigger.assert` command generates a low pulse for the programmed pulse duration.

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 8-5: RisingM output trigger



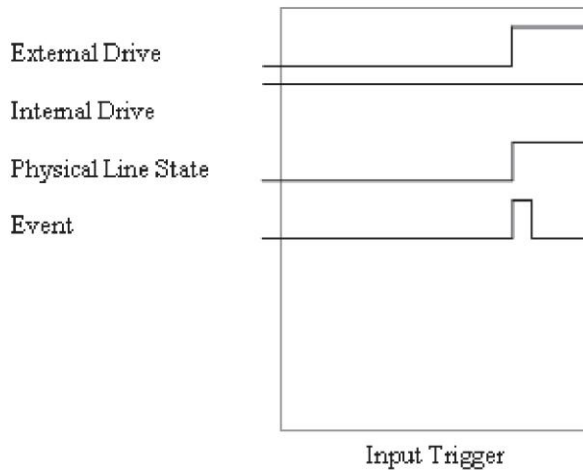
Output characteristics:

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. The following graphic displays the RisingA input trigger.

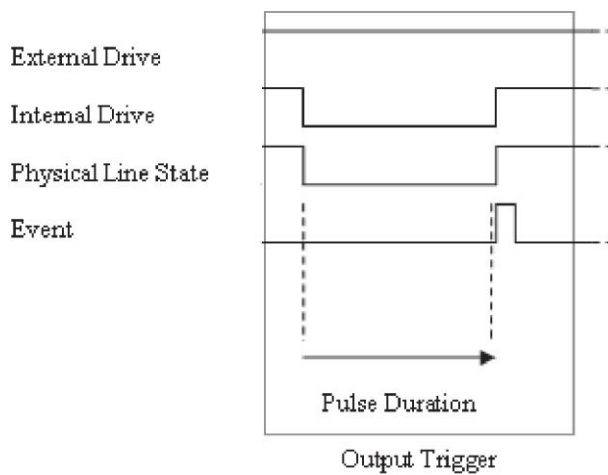
Figure 8-6: RisingA input trigger



Input characteristics:

All rising edges generate an input event.

Figure 8-7: RisingA output trigger



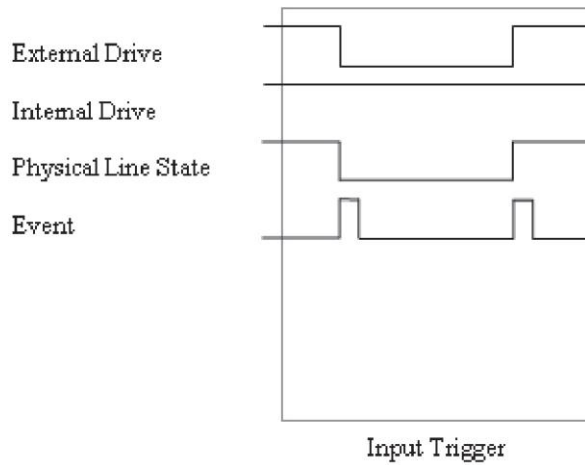
Output characteristics:

The `trigger.assert` command generates a low pulse that is similar to the falling edge trigger mode.

Either edge trigger mode

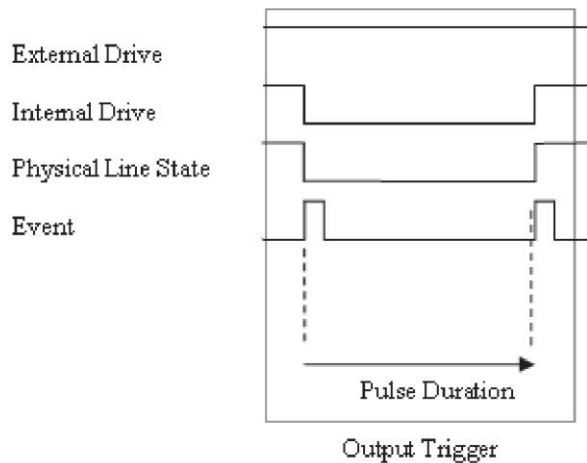
The either edge trigger mode generates a low pulse and detects both rising and falling edges.

Figure 8-8: Either edge input trigger



Input characteristics:

All rising or falling edges generate an input trigger event

Figure 8-9: 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 the 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

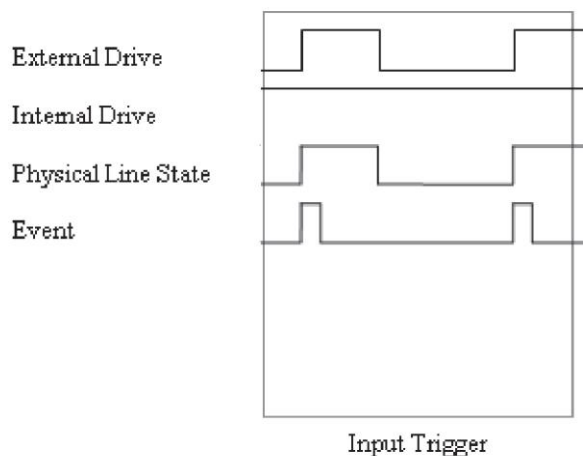
Use the synchronous master trigger mode (SynchronousM) to generate falling edge output triggers, to detect the rising edge input triggers, and to initiate an action on one or more external nodes with the same trigger line.

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.

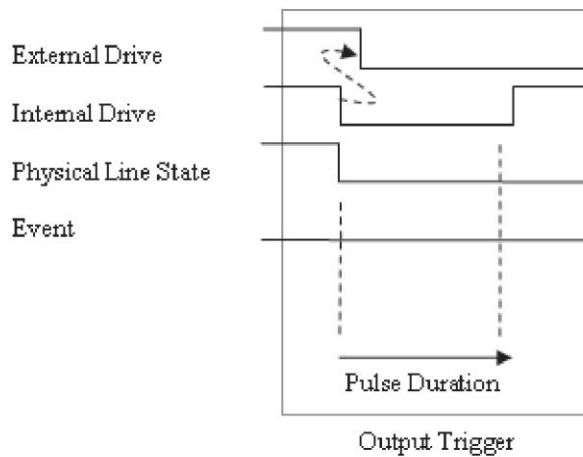
NOTE Use the SynchronousM trigger mode to receive notification when the triggered action on all nodes is complete.

Figure 8-10: 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.

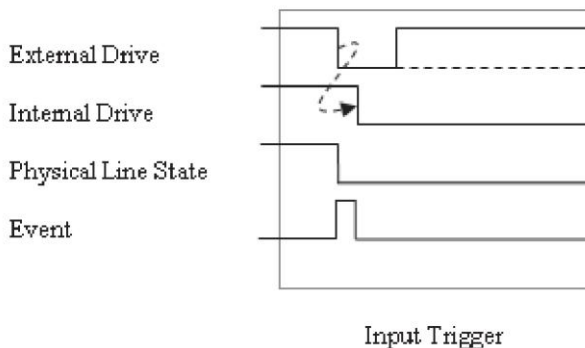
Figure 8-11: SynchronousM output trigger**Output characteristics:**

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.

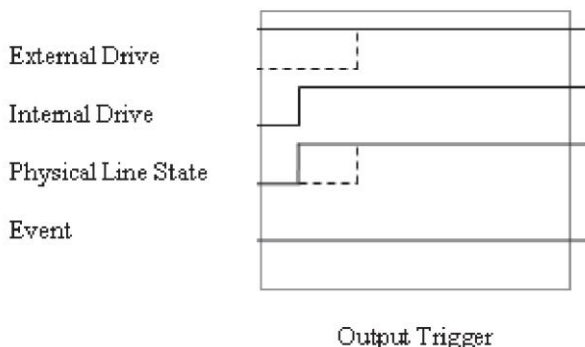
Figure 8-12: SynchronousA input trigger



Input characteristics:

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

Figure 8-13: SynchronousA output trigger



Output characteristics:

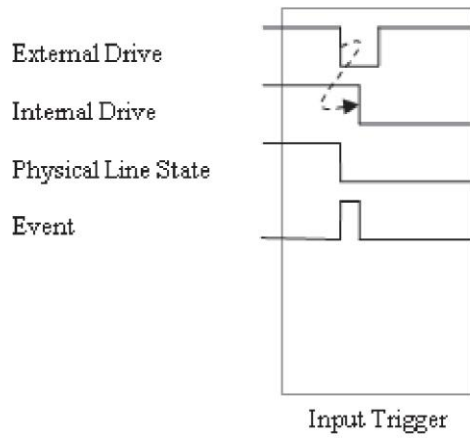
- The `trigger.assert` command releases the line if the line is latched low.
- The physical line state does not change until all drives (internal and external) release the line.

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.

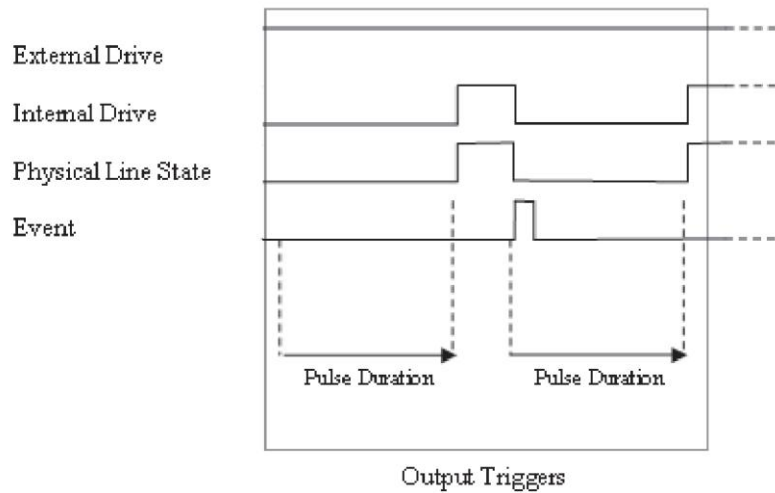
The SynchronousA and SynchronousM trigger modes provide additional flexibility. It is recommended that you use SynchronousA and SynchronousM for firmware v1.4.0 or higher, and use the synchronous trigger mode for firmware prior to v1.4.0.

Figure 8-14: Synchronous input trigger



Input characteristics:

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

Figure 8-15: 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.
- When the `trigger.assert` command is issued and the line is latched low, the pulse duration is enforced, and then the internal line drive is released.
- A normal falling edge pulse generates when the internal drive is not latched low and the `trigger.assert` command is issued.

In this section:

File formats	9-1
Default file extensions	9-1
File system navigation.....	9-2
File I/O	9-3
Script examples	9-4
Command table entries	9-9

File formats

Each script, reading buffer, and saved setup is represented on a flash drive as a separate file.

Directories on a flash drive used with the Series 3700 can only contain a limited number of files. The top-level directory is limited to approximately 150 files, while subdirectories are limited to approximately 500 files. Once the limit has been reached, a "file system full" error message is generated.

Default file extensions

You must specify the full filename, including the extension, when sending commands. Note, however, that the front panel automatically generates a generic filename that you can use as a base for naming your files. Also, some commands (for example, `io.open` (on page 9-13)) will work with either a relative or absolute path to the current working directory.

The Series 3700 has the following set of default extensions:

- **.tsp** (Test Script Processor for scripts)
- **.csv** (Comma Separated Values) for reading buffers
- **.set** for saved setups

File system navigation

The Lua FS library provides the command set necessary to navigate the file system and list the available files on a flash drive. The instrument encapsulates this command set as an `fs` logical instrument, so that the file system of any given node is available to the entire TSP-Link™ system. For example, the command `node[5].fs.readdir(".")` can be used to read the contents of the current working directory on Node 5.

To allow for future enhancements, the root folder of the USB memory stick has the absolute path `"/usb1/"`.

NOTE Both slash (/) and backslash (\) are supported as directory separators, but because backslash is an escape character in Lua, it appears as a double backslash in this context.

The following Lua FS commands, which support basic navigation and directory listing, are included for your reference.

- `fs.chdir()` (on page 9-9)
- `fs.cwd()` (on page 9-9)
- `fs.is_dir()` (on page 9-9)
- `fs.is_file()` (on page 9-9)
- `fs.mkdir()` (on page 9-9)
- `fs.readdir()` (on page 9-9)
- `fs.rmdir()` (on page 9-10)

The following Lua FS commands are not supported at this time:

- `fs.chmod`
- `fs.chown`
- `fs.stat`

File I/O

Lua supports file I/O with its `io` library commands. A subset of these commands is supported for use with Series 3700 instruments. As with Lua FS, these commands are encapsulated as an `io` logical instrument so that the files on any given node are accessible to the entire TSP-Link™ system.

Lua organizes its file I/O commands into two groups:

- Commands that reside in the `io` table, for example: `io.open`, `io.close`, `io.input`, and `io.output`. These commands are responsible for opening and closing file descriptors and performing basic I/O operations on a pair of default files, one input and one output.
- Commands that reside in the file descriptors themselves, for example: `file:seek`, `file:write`, and `file:read`. These commands operate exclusively on the file with which they are associated.

NOTE File descriptor commands for file I/O use a colon (:) to separate the command parts rather than a period (.) like the `io` commands.

A file descriptor (similar to a reading buffer) can only be used with the logical instrument that created it. Therefore, file descriptors cannot be passed between nodes in a TSP-Link system.

The default input and output files mentioned above allow for the execution of many file I/O operations without any reference to a file descriptor. Remote access to the `io.open` command is not allowed.

The following Lua I/O commands, which support basic file I/O, are included for your reference:

- `file:close()` (on page 9-10)
- `file:flush()` (on page 9-10)
- `file:read()` (on page 9-10)
- `file:seek()` (on page 9-11)
- `file:write()` (on page 9-11)
- `io.close()` (on page 9-12)
- `io.flush()` (on page 9-12)
- `io.input()` (on page 9-12)
- `io.open()` (on page 9-13)
- `io.output()` (on page 9-13)
- `io:read()` (on page 9-13)
- `io:write()` (on page 9-14)

- `io.type` (on page 9-14) (supported on the local node only when the Series 3700 is a master)

The following standard I/O commands are not supported at this time:

- `file:lines`
- `file:setvbuf`
- `io.lines`
- `io.popen`
- `io.tmpfile`

Script examples

The following script will open three different files to help illustrate the differences between the `io` commands and file descriptor commands. After opening the files, the script designates each one as the default output file (using the `io.output` command). While each file is the default for file writes (using the `io.write` command), the script also uses the file descriptor from the `io.open` to write to the file (`file:write` command).

After all files are closed (using the `io.close` command), the script will open the files again for reading. Two files are read by:

- Designating the file the default input file (using the `io.input` command)
- Being the default read contents of file (using the `io.read` command)

The third file is read by using the file descriptor from the `open` (`file:read` command). After reading all files, they are closed using the file descriptor and `close` option (`file:close` command).

```
loadscript file_io_test
-- get the current date and time
date_time = os.date('%c', os.time())
-- open the three files for writing
myfile1, myfile1_err, myfile1_errnum =
  io.open('/usb1/myfile_io1', 'w')
myfile2, myfile2_err, myfile2_errnum =
  io.open('/usb1/myfile_io2', 'w')
myfile3, myfile3_err, myfile3_errnum =
  io.open('/usb1/myfile_io3', 'w')
if (io.type(myfile1) == 'file') then
  if (io.type(myfile2) == 'file') then
    if (io.type(myfile3) == 'file') then
      -- make myfile1 the default output file
      io.output(myfile1)
      -- write some data to the default file
      io.write('Using io write to myfile1 to io output\n')
      io.write(date_time)
      io.write('\n')
      -- now write to myfile2 using descriptor rather than
      io write command
      myfile2:write('  file handle to write to
myfile2\n')
      myfile2:write('  while myfile1 is output file for
io\n')
      -- make myfile2 the default output file
      io.output(myfile2)
      -- write some data to the default file
      io.write('Using io write to myfile2 to io output\n')
      io.write(date_time)
      io.write('\n')
      -- now write to myfile3 using descriptor rather than
      io write command
      myfile3:write('  file handle to write to
myfile3\n')
      myfile3:write('  while myfile2 is output file for
io\n')
      -- make myfile3 the default output file
      io.output(myfile3)
      -- write some data to the default file
      io.write('Using io write to myfile3 to io output\n')
      io.write(date_time)
      io.write('\n')
      -- now write to myfile1 using descriptor rather than
      io write command
      myfile1:write('  file handle to write to
myfile1\n')
      myfile1:write('  while myfile3 is output file for
io\n')
      -- use the io close rather than file descriptor
      close command
      io.close(myfile1)
      io.close(myfile2)
      io.close(myfile3)
    else
      print('myfile3 did not open for write')
      print('error string is ' .. myfile3_err)
```

```

        print('error number is ' .. myfile3_errnum)
    end
else
    print('myfile2 did not open for write')
    print('error string is ' .. myfile2_err)
    print('error number is ' .. myfile2_errnum)
end
else
    print('myfile1 did not open for write')
    print('error string is ' .. myfile1_err)
    print('error number is ' .. myfile1_errnum)
end
-- open the 3 files again for reading
myfile1, myfile1_err, myfile1_errnum =
    io.open('/usb1/myfile_io1', 'r')
myfile2, myfile2_err, myfile2_errnum =
    io.open('/usb1/myfile_io2', 'r')
myfile3, myfile3_err, myfile3_errnum =
    io.open('/usb1/myfile_io3', 'r')
if (io.type(myfile1) == 'file') then
    if (io.type(myfile2) == 'file') then
        if (io.type(myfile3) == 'file') then
            -- make myfile1 the default input file
            io.input(myfile1)
            -- read the default file
            filecontents = io.read('*a')
            print('contents of myfile1 are:')
            print(filecontents)
            print()
            -- make myfile2 the default input file
            io.input(myfile2)
            -- read the default file
            filecontents = io.read('*a')
            print('contents of myfile2 are:')
            print(filecontents)
            print()
            -- read myfile3 using file descriptor instead of io
            read
            filecontents = myfile3:read('*a')
            print('contents of myfile3 are:')
            print(filecontents)
            print()
            -- use file descriptor close command rather than io
            close
            myfile1:close()
            myfile2:close()
            myfile3:close()
        else
            print('myfile3 did not open for read')
            print('error string is ' .. myfile3_err)
            print('error number is ' .. myfile3_errnum)
        end
    end
else
    print('myfile2 did not open for read')

```



```
        print('error string is ' .. myfile2_err)
        print('error number is ' .. myfile2_errnum)
    end
else
    print('myfile1 did not open for read')
    print('error string is ' .. myfile1_err)
    print('error number is ' .. myfile1_errnum)
end
endscript
```

After downloading the above script, type `file_io_test()` to execute the script:

```
file_io_test()
```

The following output is returned after executing the `file_io_test()` script:

```
contents of myfile1 are:
Using io write to myfile1 to io output
11/27/07 07:57:23
    file handle to write to myfile1
    while myfile3 is output file for io
```

```
contents of myfile2 are:
    file handle to write to myfile2
    while myfile1 is output file for io
Using io write to myfile2 to io output
11/27/07 07:57:23
```

```
contents of myfile3 are:
    file handle to write to myfile3
    while myfile2 is output file for io
Using io write to myfile3 to io output
11/27/07 07:57:23
```

The following script will open a file called `myfiletest` three times. The first time it is opened is for writing. Note that opening an existing file for writing deletes any existing information in the file. The second time it is opened is for appending more data to the existing data in the file. Opening a file for append will not delete any existing data; it only adds data to the end of the existing file contents. The third time the file is opened is for reading the entire contents of the file (existing data and appended data).

```
loadscript filetest
-- script to write 2 lines to a file
-- append 2 lines to the same file
-- read the entire file contents and print them

-- open the file for writing
myfile = io.open('/usb1/myfiletest', 'w')
if io.type(myfile) == 'file' then
    myfile:write('This is my first line WRITING\n')
    myfile:write('This is my next line WRITING\n')
    myfile:close()

-- open the file for appending
myfile = io.open('/usb1/myfiletest', 'a')
if io.type(myfile) == 'file' then
    myfile:write('This is my first APPEND line\n')
    myfile:write('This is my next APPEND line\n')
    myfile:close()

-- open the file for reading
myfile = io.open('/usb1/myfiletest', 'r')
if io.type(myfile) == 'file' then
    filecontents = myfile:read('*a')
    print('the file contains:')
    print()
    print(filecontents)
    myfile:close()
else
    print('The file did not open correctly for reading')
end
else
    print('The file did not open correctly for appending')
end
else
    print('The file did not open correctly for writing')
end
endscript
```

After downloading the above script, type `filetest()` to execute the script.
Here are the output results:

```
the file contains:
This is my first line WRITING
This is my next line WRITING
This is my first APPEND line
This is my next APPEND line
```

Command table entries

fs.chdir()	
Function	Sets the current working directory.
Usage	<code>fs.chdir(path)</code> path: The new working directory path (absolute or relative).
Remarks	An error is logged to the error queue if the given path does not exist.
fs.cwd()	
Function	Returns the absolute path of the current working directory.
Usage	<code>path = fs.cwd()</code> path: The absolute path of the current working directory.
fs.is_dir()	
Function	Tests whether the specified path refers to a directory.
Usage	<code>status = fs.is_dir(path)</code> status: True if the given path is a directory; otherwise, false. path: The file system entry path (absolute or relative) to test.
Remarks	An error is logged to the error queue if the given path does not exist.
fs.is_file()	
Function	Tests whether the specified path refers to a file (as opposed to a directory).
Usage	<code>status = fs.is_file(path)</code> status: True if the given path is a file; otherwise, false. path: The path of the file system entry to test. This path may be absolute or relative to the current working directory.
Remarks	An error is logged to the error queue if the given path does not exist.
fs.mkdir()	
Function	Creates a directory at the specified path.
Usage	<code>fs.mkdir(path)</code> path: The path of the new directory. This path may be absolute or relative to the current working directory.
Remarks	An error is logged to the error queue if the parent folder of the new directory does not exist, or if a file system entry already exists at the given path.

fs.readdir()	
Function	Returns a list of all the file system entries within a specified directory.
Usage	<pre>files = fs.readdir(path)</pre> <p>files: A list containing the names of all the file system entries that reside in the specified directory.</p> <p>path: The directory path. This path may be absolute or relative to the current working directory.</p>
Remarks	This command is non-recursive (that is, entries in subfolders are not returned). An error is logged to the error queue if the given path does not exist, or does not represent a directory.

fs.rmdir()	
Function	Removes a directory from the file system.
Usage	<pre>fs.rmdir(path)</pre> <p>path: The path of the directory to remove. This path may be absolute or relative to the current working directory.</p>
Remarks	An error is logged to the error queue if the given path does not exist, does not represent a directory, or if the directory is not empty.

file:close()	
Function	Closes a file after flushing any data that was written to it with <code>io.write()</code> (on page 9-14) or <code>file:write()</code> (on page 9-11).
Usage	<pre>file:close()</pre> <p>file: The descriptor of the file to close.</p>
Remarks	This command is equivalent to <code>io.close(file)</code> . It is not remotely accessible.

file:flush()	
Function	Flush the buffered data for the specified file
Usage	<pre>file:flush()</pre> <p>file: The descriptor of the file to flush</p>
Remarks	Use this command to flush data written to it by <code>file:write()</code> (on page 9-11) or <code>io.write()</code> (on page 9-14). Using this function removes the need to close a file after writing to it and allows it to be left open to write more data. Data may be lost if the file is not closed or flushed before an application ends. To prevent the loss of data if there is going to be a time delay before more data is written when you want to keep file open and not close it, flush the file after writing to it.

file:read()	
Function	Reads data from a file.
Usage	<pre>data = file:read(format)</pre> <p>data: The data read from the file. The number of return values matches the number of values in <i>format</i>.</p> <p>file: The descriptor of the file to read.</p> <p>format: A string or number indicating the type of data to be read. Any number of format parameters may be passed to this command, each corresponding to a returned data value. The format attribute is optional; the default is "*".</p>
Remarks	<p>The <i>format</i> parameters may be any of the following:</p> <p>"*n": Return a number.</p> <p>"*a": Return the whole file, starting at the current position; return the empty string at the end of the file.</p> <p>"*l": Return the next line, skipping the end of line; return <i>nil</i> at the end of file.</p> <p><i>n</i>: Return a string with up to <i>n</i> characters; return an empty string if <i>n</i> is zero; return <i>nil</i> at the end of file.</p> <p>Any error encountered is logged to the error queue.</p> <p>This command is not remotely accessible.</p>
file:seek()	
Function	Sets and gets a file's current position.
Usage	<pre>position = file:seek(whence, offset)</pre> <p>position: The new file position, measured in bytes from the beginning of the file.</p> <p>file: The descriptor of the file.</p> <p>whence: A string indicating the base against which offset is applied. The whence attribute is optional; the default is "cur".</p> <p>offset: The intended new position, measured in bytes from a base indicated by whence. Optional, default is 0.</p>
Remarks	<p>The <i>whence</i> parameters may be any of the following:</p> <p>"set": Beginning of file.</p> <p>"cur": Current position.</p> <p>"end": End of file.</p> <p>If an error is encountered, it is logged to the error queue, and the command returns <i>nil</i> and the error string.</p> <p>This command is not remotely accessible.</p>

file.write()	
Function	Buffer data until a flush (<code>file:flush()</code> (on page 9-10) or <code>io.flush()</code> (on page 9-12)) or close (<code>file:close()</code> (on page 9-10) or <code>io.close()</code> (on page 9-12)) operation is performed.
	NOTE Data may be lost if the file is not flushed or closed before the application ends. A write function buffers the data until a flush or close operation is requested.
Usage	<code>file:write(data)</code> file: The descriptor of the file. data: The data to write to the file. An arbitrary number of data values may be passed to this command. All parameters must be either strings or numbers.
Remarks	Any error encountered is logged to the error queue. This command is not remotely accessible.

io.close()	
Function	Closes the specified file.
Usage	<code>io.close(file)</code> file: A file descriptor to flush and close
Remarks	This command is equivalent to <code>file:close()</code> (on page 9-10).

io.flush()	
Function	Flush the buffered data for the current output file.
Usage	<code>io.flush()</code>
Remarks	Use this command to flush data written to the current default file by <code>file:write()</code> (on page 9-11) or <code>io.write()</code> (on page 9-14). Using this command removes the need to close a file after writing to it and allows it to be left open to write more data. Data may be lost if the file is not closed or flushed before an application ends. To prevent the loss of data if there is going to be a time delay before more data is written when you want to keep file open and not close it, flush the file after writing to it.

io.input()	
Function	Assigns a previously opened file, or opens a new file, as the default input file.
Usage	<code>io.input(filein)</code> <code>fileout = io.input()</code> filein: A file descriptor to assign or the path of a file to open as the default input file. The path may be absolute or relative to the current working directory. This parameter is optional; if absent, the command returns the absolute path to the current default input file (<code>fileout</code>). fileout: The absolute path to the default input file.

io.input()	
Remarks	Any error encountered is logged to the error queue. The remotely-accessible version of this command does not accept a file descriptor parameter.

io.open()	
Function	Opens a file for later access.
Usage	<code>file, err, errnum = io.open(path, mode)</code> file : The descriptor of the opened file. err : A string with an error message an error occurred. errnum : Number representing the error number. path : The path of the file to open. This path may be absolute or relative to the current working directory. mode : A string representing the intended access mode. The mode attribute is optional; the default is "r".
Remarks	The <code>mode</code> string can be any of the standard C language <code>fopen</code> modes, including: "r": Read mode. "w": Write mode. "a": Append mode. If an error is encountered, it is logged to the error queue, and the command returns <code>nil</code> and the error string. This command is not remotely accessible.

io.output()	
Function	Assigns a previously opened file or opens a new file as the default output file.
Usage	<code>io.output(filein)</code> <code>fileout = io.output()</code> filein : A file descriptor to assign, or the path of a file to open, as the default output file. The path may be absolute or relative to the current working directory. This parameter is optional; if absent, the command returns the absolute path to the current default output file (<code>fileout</code>). fileout : The absolute path to the default output file.
Remarks	Any error encountered is logged to the error queue. The remotely-accessible version of this command does not accept a file descriptor parameter.

io.read()	
Function	Reads data from the default input file.
Usage	<pre>data = io.read(format)</pre> <p>data: The data read from the file. The number of return values matches the number of values in <code>format</code>.</p> <p>format: A string or number indicating the type of data to be read. Any number of format parameters may be passed to this command, each corresponding to a returned data value. Optional; default is <code>"l"</code>.</p>
Remarks	<p>The <code>format</code> parameters may be any of the following:</p> <p>"*n": Return a number.</p> <p>"*a": Return the whole file, starting at the current position; return an empty string at the end of file.</p> <p>"*l": Return the next line, skipping the end of line; return <code>nil</code> at the end of file.</p> <p>n: Return a string with up to <code>n</code> characters; return an empty string if <code>n</code> is zero; return <code>nil</code> at the end of file.</p> <p>Any error encountered is logged to the error queue.</p>

io.type()	
Function	Checks whether <code>obj</code> is a valid file handle.
Usage	<code>io.type(obj)</code>
Remarks	Returns "file" if <code>obj</code> is an open file handle, "closed file" if <code>obj</code> is a closed file handle, and <code>nil</code> if <code>obj</code> is not a file handle.

io.write()	
Function	<p>Buffer data until a flush (<code>file:flush()</code> (on page 9-10) or <code>io.flush()</code> (on page 9-12)) or close (<code>file:close()</code> (on page 9-10) or <code>io.close()</code> (on page 9-12)) operation is performed.</p> <hr/> <p>NOTE Data may be lost if the file is not flushed or closed before the application ends. A write buffers the data until a flush or close operation is requested.</p>
Usage	<pre>io.write(data)</pre> <p>data: The data to write to the file. An arbitrary number of data values may be passed to this command. All parameters must be either strings or numbers.</p>
Remarks	Any error encountered is logged to the error queue.

In this section:

Overview	10-1
TSP-Net™ Capabilities.....	10-1
Using TSP-Net™ with any Ethernet-enabled device	10-2
Using TSP-Net™ vs. TSP-Link™ for communication with TSP-enabled devices	10-4
Instrument Control Library (ICL) - General device control	10-5
Instrument Control Library - TSP-specific device control	10-12

Overview

TSP-Net™ allows the Series 3700 to control Ethernet-enabled devices directly through its LAN port. This enables the Series 3700 to communicate directly with a non-TSP™-enabled device without the use of a controlling computer.

TSP-Net™ Capabilities

For both TSP™ and non-TSP devices, the TSP-Net library permits the Series 3700 to control a remote device through the LAN port. Using TSP-Net methods, you can transfer string data to and from a remote device, transfer and format data into Lua variables, and clear input buffers. TSP-Net is only accessible using ICL commands from a remote command interface and is not available from the front panel.

You can use TSP-Net to communicate with any Ethernet-enabled device. However, specific TSP-Net commands exist for TSP-enabled devices to allow for support of features unique to TSP. These features include script downloads, reading buffer access, wait completion, and handling of TSP prompts.

Using TSP-Net with TSP-enabled instruments, a Series 3700 can download a script to another TSP-enabled device and have both devices run scripts independently. The Series 3700 can read the data from the remote device and either manipulate the data or send the data to a different remote device on the LAN. You can simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the Series 3700.

Using TSP-Net™ with any Ethernet-enabled device

NOTE Refer to the *Instrument Control Library (ICL)* (on page 12-1) for more details on the commands presented in this section.

To communicate to a remote Ethernet-enabled device from the Series 3700, perform the following steps:

1. Connect to the remote device through the LAN port.

Use an Ethernet crossover cable to connect directly from the Series 3700 to an Ethernet-enabled device.

Use a straight-through Ethernet cable and a hub to connect the Series 3700 to any other device on the LAN.

2. Establish a new connection to a remote device at a specific IP address using `tspnet.connect()` (on page 10-5). For non-TSP™-enabled devices, you must also provide the port number, or the Series 3700 assumes the remote device to be TSP-capable and enables TSP prompts and error handling.

If the Series 3700 is not able to make a connection to the remote device, it generates a timeout error. Use `tsp.timeout` to set the timeout value. The default timeout value is 20 seconds.

NOTE Set `tspnet.tsp.abortonconnect` (on page 10-14) to TRUE to abort any script currently running on a remote TSP device.

3. Use `tspnet.write()` (on page 10-7) or `tspnet.execute()` (on page 10-6) to send strings to a remote device. Using `tspnet.write()` sends strings to the device exactly as indicated, and you must supply any needed termination characters or other lines. Use `tspnet.termination()` (on page 10-11) to specify the termination character. If you use `tspnet.execute()` (on page 10-6) instead, the Series 3700 appends termination characters to all strings sent to the command.
4. Retrieve responses from the remote device using `tspnet.read()` (on page 10-8). The Series 3700 suspends operation until data is available or a timeout error is generated. You can check if data is available from the remote device using `tspnet.readavailable()` (on page 10-9).

Disconnect from the remote device using `tspnet.disconnect()` (on page 10-10). Terminate all remote connections using `tspnet.reset()` (on page 10-10).

Example script

The following example demonstrates how to connect to a remote non-TSP™-enabled device, and send and receive data from this device:

```
-- Disconnect all existing TSP-Net™ connections.
tspnet.reset()

-- Set tspnet timeout to 5 seconds.
tspnet.timeout = 5

-- Establish connection to another device with IP address
  192.168.1.51 at port 1394.
id_instr = tspnet.connect("192.168.1.51",1394, "*rst\r\n")

-- Print the device ID from connect string.
print("ID is: ", id_instr)

-- Set termination character to CRLF. You must do this on a
  per connection basis after connection has been made.
tspnet.termination(id_instr, tspnet.TERM_CRLF)

-- Send the command string to the connected device
tspnet.write(id_instr,"*idn?" .. "\r\n")

-- Read the data available, then prints it.
print("instrument write/read returns:: " ,
      tspnet.read(id_instr))

-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

Using TSP-Net™ vs. TSP-Link™ for communication with TSP-enabled devices

TSP-Link is the preferred communication method when communicating between the Series 3700 and another TSP™-enabled instrument. Using TSP-Link has certain advantages over using TSP-Net, including:

- **Error checking:** When connected to a TSP-enabled device, all errors that occur on the remote device are transferred to the error queue of the Series 3700. The Series 3700 indicates errors from the remote device by prefacing these errors with “Remote Error”.

For example, if the remote device generates error number 4909, the Series 3700 generates the error string “Remote Error: (4909) Reading buffer not found within device.”

- **Digital I/O Triggering:** TSP-Link connections have three TSP synchronization lines that are available to each device on the TSP-Link network. You can use any one of the TSP synchronization lines to perform hardware triggering between devices on the TSP-Link network. Refer to [Hardware trigger modes](#) (on page 8-18) for more details.

These advantages make using TSP-Link to control another TSP-enabled device the best choice for most applications. However, if the distance between the Series 3700 and the TSP-enabled device is longer than 15 feet, use TSP-Net.

To establish a remote TSP-Net connection with a TSP-enabled device, use `tsp.connect()` without specifying a port number. The Series 3700 enables TSP prompt and error handling for the remote device, which allows you to successfully use the `tspnet.tsp` set of commands to load and run scripts and retrieve reading buffers.

Abort any operation on the remote TSP-enabled device using `abort()`.

Instrument Control Library (ICL) - General device control

tspnet.connect()	
Function	Device connection.
Usage	<p>To connect to any remote device on the LAN:</p> <pre><connection id> = tspnet.connect([<ip address>, [<port number>, <initialize string>]])</pre> <p>To connect to a TSP-enabled remote device on the LAN:</p> <pre><connection id> = tspnet.connect([<ip address>, [<password>]])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>ip address: String variable for passing the IP address</p> <p>port number: Optional integer value of the port number</p> <p>initialize string: String type for the initialization string to send</p>
Remarks	<p>This command connects a device to another device by way of the LAN interface (using the optionally-specified port number). The default port number is 5025. If the port number is 23, the interface will use the Telnet protocol (and set appropriate termination characters) to communicate with the device.</p> <p>If a port number and initialization string are provided, the remote device is assumed to be non-TSP-enabled. The Series 3700 does not perform any extra processing, prompt handling, error handling, or sending of commands. Additionally, the <code>tspnet.tsp</code> commands do not apply for use on this this remote device.</p> <p>If no port number and initialization string is provided, the remote device is assumed to be a Keithley Instruments TSP-enabled device. Depending on the state of <code>tspnet.tsp.abortonconnect</code> (on page 10-14), the Series 3700 sends an <code>abort()</code> to the remote device upon connection. The Series 3700 also enables TSP prompts on the remote device and error management. The Series 3700 places remote errors from the TSP-enabled device in its own error queue and prefaces these errors with "Remote Error", followed by an error description. Do not manually change either the prompt functionality (<code>localnode.prompts</code>) or show errors functionality (<code>localnode.showerrors</code>) on the remote TSP-enabled device, or subsequent <code>tspnet.tsp.*</code> commands using the connection may fail.</p> <p>You can simultaneous connect to a maximum of 32 remote devices.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Connection Failed • Connection Failed, Timeout • Invalid IP Address or Port Number

tspnet.connect()	
Example	<p>To connect to a TSP-enabled device:</p> <pre>mytspdevice = tspnet.connect('10.80.64.216')</pre> <p>To connect to a non-TSP-enabled device:</p> <pre>mydevice = tspnet.connect("192.168.1.51",1394, "*rst\r\n")</pre>

tspnet.idn()	
Function	Retrieves response of remote device to '*IDN?'
Usage	<pre><idn string> = tspnet.idn(<connection id>)</pre> <p>idn_string: Response as a string type</p> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p>
Remarks	<p>Sends the '*idn?' string to the remote device and retrieves its response.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Connection Not Available • Connection Failed, Aborted • Write Failed, Timeout • Write Failed • Read Failed, Timeout • Read Failed • Read Failed, Aborted
Example	<p>Retrieve and print response of 'IDN?*' from the remote device:</p> <pre>print(tspnet.idn(mydevice))</pre> <p>KEITHLEY INSTRUMENTS INC.,MODEL 3706,34345656,01.02a</p>

tspnet.execute()	
Function	Executes a command string on the remote device.
Usage	<pre>[variable =] tspnet.execute(<connection id>, <command string>, [<format string>])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>command string: Command to send to instrument.</p> <p>format string: Definition of format string for the input field using zeros (0), the decimal point (.), the polarity sign (+), and 'E' for exponent.</p>

tspnet.execute()	
Remarks	<p>This command sends the command string to the connection device. A termination is added to the command string when it is sent to the device (see <code>tspnet.termination()</code> (on page 10-11)). Optionally, when a format string is specified, the command waits for a string from the device. The Series 3700 decodes the output string according to the format specified in the format string and returns this output string as arguments from the function (see <code>tspnet.read()</code> (on page 10-8) for format specifiers).</p> <p>When this command is sent to a TSP-enabled device, the Series 3700 suspends operation until a timeout error is generated or until the device responds, even if no format string is specified. The TSP prompt from the remote device is read and thrown away. The Series 3700 places any remotely-generated errors into its error queue. When the optional format string is not specified, this command is equivalent to <code>tspnet.write()</code> (on page 10-7), except that a termination is automatically added to the end of the line.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Write Failed, Timeout • Write Failed • Read Failed, Timeout • Read Failed, Aborted • Read Failed • Remote Error, <remote error generated by command>
Example	<pre> Command remote device to run script named 'runmyscript()': tspnet.execute(mydevice, 'runmyscript()') Command remote device to execute a *idn?: tspnet.termination(mydevice, tspnet.TERM_CRLF) tspnet.execute(mydevice, '*idn?') print("instrument write/read returns:: " , tspnet.read(id_instr)) </pre>

tspnet.write()	
Function	Write strings to remote device.
Usage	<pre>tspnet.write(<connection id>, <input string>)</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>input string: String type used for writing to the remote instrument</p>

tspnet.write()	
Remarks	<p>The <code>tspnet.write()</code> command sends the command string to the connection device. It does not wait for command completion on the remote device.</p> <p>The Series 3700 sends the input string to the remote device exactly as indicated. The input string must contain any necessary new lines, termination, or other indicators.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Write Failed, Timeout • Write Failed
Example	<p>Command remote device to run script named 'runmyscript':</p> <pre>tspnet.write(mydevice, 'runmyscript()\n')</pre> <p>Send a *idn? to a remote device:</p> <pre>tspnet.write(id_instr, "*idn?" .. "\r\n")</pre> <p>or</p> <pre>tspnet.write(id_instr, "*idn?\r\n")</pre>
tspnet.read()	
Function	Reads data from remote device.
Usage	<pre>[variable =] tspnet.read(<connection id>, [<format string>])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>format string: Definition of format string for the input field using zeros (0), the decimal point (.), the polarity sign (+), and 'E' for exponent.</p>

tspnet.read()									
Remarks	<p>This command reads available data from the device (as indicated by the format string) and returns the number of arguments (as indicated by the format string).</p> <p>The format string can contain the following identifiers:</p> <table border="0"> <tr> <td style="padding-right: 20px;">%[width]s</td> <td>Read data until the specific length</td> </tr> <tr> <td>%[max width]t</td> <td>Read data until the specific length or delimited by punctuation</td> </tr> <tr> <td>%[max width]n</td> <td>Read data until a newline and/or carriage return</td> </tr> <tr> <td>%d</td> <td>Read a number (delimited by punctuation)</td> </tr> </table> <p>If no format is specified, the command returns a string containing the data until a new line is reached. If no data is available, the Series 3700 will hold off operation until the requested data is available or until a timeout error is generated. Use <code>tspnet.timeout</code> to specify the timeout period.</p> <p>A maximum of 10 specifiers are allowed in a format string.</p> <p>When reading from a TSP-enabled remote device, the Series 3700 removes TSP prompts and places any errors received from the remote device into its own error queue. The Series 3700 prefaces errors from the remote device with "Remote Error," and followed by with the error number and error description.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Read Failed, Timeout • Read Failed, Aborted • Read Failed • Remote Error, <remote error generated by command> 	%[width]s	Read data until the specific length	%[max width]t	Read data until the specific length or delimited by punctuation	%[max width]n	Read data until a newline and/or carriage return	%d	Read a number (delimited by punctuation)
%[width]s	Read data until the specific length								
%[max width]t	Read data until the specific length or delimited by punctuation								
%[max width]n	Read data until a newline and/or carriage return								
%d	Read a number (delimited by punctuation)								
Example	<p>Send <code>"*idn?"</code> to remote device:</p> <pre>tspnet.write(id_instr, "*idn?\r\n")</pre> <p>Read and print response from remote device:</p> <pre>print("instrument write/read returns:: " , tspnet.read(id_instr))</pre>								

tspnet.readavailable()	
Function	Device read output available.
Usage	<p>[<num bytes> =] <code>tspnet.readavailable(<connection id>)</code></p> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p>
Remarks	<p>This command checks to see if any output data is available from the device. No data is read. It is intended to allow TSP™ scripts to continue to run without waiting on a remote command to finish.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Read Failed

tspnet.readavailable()	
Example	<code>x = tspnet.readavailable(mydevice)</code>

tspnet.clear()	
Function	Device read clear buffer.
Usage	<code>tspnet.clear(<connection id>)</code> connection id: Integer value used as a handle for other <code>tspnet</code> commands
Remarks	This command clears any pending output data available from the device. No data is returned to the caller. No data is processed. Errors: <ul style="list-style-type: none"> Invalid Specified Connection
Example	<code>tspnet.write(mydevice, 'print([[hello]])')</code> <code>print(tspnet.readavailable(mydevice))</code> Output 6.00000000e+000 <code>tspnet.clear(mydevice)</code> <code>print(tspnet.readavailable(mydevice))</code> Output 0.00000000e+000

tspnet.disconnect()	
Function	Device disconnection.
Usage	<code>tspnet.disconnect(<connection id>)</code> connection id: Integer value used as a handle for other <code>tspnet</code> commands
Remarks	This command disconnects the two devices by closing the connection. For Keithley Instruments TSP™ devices, this results in any remotely running commands or scripts being aborted (terminated). Errors: <ul style="list-style-type: none"> Invalid Specified Connection
Example	<code>tspnet.disconnect(mydevice)</code>

tspnet.reset()	
Function	Device all disconnection.
Usage	<code>tspnet.reset()</code>

tspnet.reset()	
Remarks	<p>This command disconnects the all devices currently connected.</p> <p>For Keithley Instruments TSP™ devices, this results in any remotely running commands or scripts being terminated.</p> <p>Errors:</p> <ul style="list-style-type: none"> • <none>
Example	<code>tspnet.reset()</code>

tspnet.termination()	
Function	Device line termination.
Usage	<pre><termination type> = tspnet.termination(<connection id>, [<termination type>])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>termination type: <code>tspnet.TERM_LF</code>, <code>tspnet.TERM_CR</code>, <code>tspnet.TERM_CRLF</code>, or <code>tspnet.TERM_LFCR</code></p>
Remarks	<p>This setting sets and gets the termination characters used to determine the end of a line for lines being received by a connection. It also is used to terminate lines being sent to a connection. Pass the optional set value to set the termination. The current value is always returned. There are four possible values: LF, CR, CRLF, or LFCR. For TSP™ devices, the default is LF. For non-TSP devices, the default is CRLF. The termination character resets to default when a connection is terminated.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Invalid Termination
Example	<p>Set termination character:</p> <pre>tspnet.termination(mydevice, tspnet.TERM_LF)</pre> <p>Gets termination character and evaluates if set to LF. Response of "1" means true, set to <termination type>. Response of "0" means false, not set to <termination type>:</p> <pre>print(tspnet.termination(mydevice) == tspnet.TERM_LF)</pre> <p>Output:</p> <pre>1.0000000e+000</pre>

tspnet.timeout	
Attribute	Sets timeout value for <code>tspnet.connect()</code> , <code>tspnet.execute()</code> , and <code>tspnet.read()</code> commands.
Usage	<pre>tspnet.timeout [= <seconds value>]</pre> <p>seconds value: Value in seconds</p>

tspnet.timeout	
Remarks	<p>This setting sets the duration the <code>tspnet.connect</code>, <code>tspnet.read</code>, and <code>tspnet.execute</code> commands will wait for a response. The time is specified in seconds. The default value is 5.0 seconds. The timeout may contain fractional seconds but is only accurate to the nearest 10mS. The timeout may be between 0.0 and 30 seconds.</p> <p>Errors:</p> <ul style="list-style-type: none"> Invalid Timeout
Example	<code>tspnet.timeout = 10.0</code>

Instrument Control Library - TSP-specific device control

tspnet.tsp.runscript()	
Function	Load and runs a script on a device.
Usage	<pre>tspnet.tsp.runscript(<connection id>, [<name>,< >] <script>)</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>name: Optional parameter name from a listed group</p> <p>script: The actual script itself as a string, enclosed in quotes</p>
Remarks	<p>This convenience command downloads a script to a device and runs it. It automatically adds the appropriate <code>loadscript</code> and <code>endscript</code> around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.</p> <p>The script is automatically loaded, compiled, and run. If there are no runnable lines (contains only functions), running has no effect.</p> <p>To load only and run at a later time, simply make sure the script contains only functions. Use <code>tspnet.execute()</code> to execute those functions at a later time.</p> <p>This command is appropriate only for TSP™-enabled devices.</p> <p>If no name is specified, one will be generated internally.</p> <p>Errors:</p> <ul style="list-style-type: none"> Invalid Specified Connection Write Failed, Timeout Write Failed Read Failed, Timeout Read Failed, Aborted Read Failed Remote Error, <remote error generated by command>
Example	<pre>tspnet.tsp.runscript(mytspdevice, 'mytest', 'print([[start]]) for d = 1,10 do print([[work]]) end print([[end]])')</pre>

tspnet.tsp.rhtablecopy()	
Function	Copies a reading buffer synchronous table from a device.
Usage	<pre><array> = tspnet.tsp.rhtablecopy(<connection id>, <name>, [<start index>, <end index>])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>name: Parameter name from a listed group</p> <p>start index: Integer start value</p> <p>end index: Integer end value</p>
Remarks	<p>This convenience command reads the data from a reading buffer on a remote device and returns an array of numbers or a string representing the data. The name argument identifies the reading buffer name and synchronous table to copy. The optional start index and end index specify the portion of the reading buffer to read. If no index is specified, the entire buffer will be copied.</p> <p>This command is limited to transferring 50,000 readings at a time.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Write Failed, Timeout • Write Failed • Read Failed, Timeout • Read Failed, Aborted • Read Failed • Invalid Reading Buffer Table • Invalid Index Range • Out of Memory • Remote Error, <remote error generated by command>
Example	<pre>table = tspnet.tsp.rhtablecopy(mytspdevice, 'myremotebuffername.readings', 1, 3) print(table[1], table[2], table[3])</pre> <p>Output:</p> <pre>4.5653423423e-1 4.5267523423e-1 4.5753543423e-1</pre> <pre>times = tspnet.tsp.rhtablecopy(mytspdevice, 'myremotebuffername.timestamps', 1, 3) print(times)</pre> <p>Output</p> <pre>01/01/2008 10:10:10.0000013,01/01/2008 10:10:10.0000233,01/01/2008 10:10:10.0000576</pre>

tspnet.tsp.abort()	
Function	Aborts device execution.
Usage	<code>tspnet.tsp.abort(<connection id>)</code> connection id: Integer value used as a handle for other <code>tspnet</code> commands
Remarks	This convenience command simply sends an "abort" string to a device. Errors: <ul style="list-style-type: none"> • Invalid Specified Connection • Connection Not Available • Write Failed
Example	<code>tspnet.tsp.abort()</code>

tspnet.tsp.abortonconnect	
Attribute	Abort on connect.
Usage	<code>tspnet.tsp.abortonconnect [= <value>]</code> value: <code>tspnet.TRUE</code> or <code>tspnet.FALSE</code>
Remarks	This setting determines if the Series 3700 sends <code>abort()</code> when it attempts to connect using <code>tspnet.connect()</code> (on page 10-5) to a TSP™-enabled device. The default value is <code>tspnet.TRUE</code> (or non-zero). Sending the <code>abort()</code> command on connection causes any other active interfaces being used on that device to close to ensure you have obtained access to the remote device. Connecting to a TSP device without issuing an <code>abort()</code> command, or when <code>tspnet.tsp.abortonconnect</code> (on page 10-14) is set to <code>tspnet.FALSE</code> , can result in the Series 3700 suspending operation until it receives a response back from the device or until a timeout error generates. Errors: <ul style="list-style-type: none"> • <none>
Example	<code>tspnet.tsp.abortonconnect = tspnet.FALSE</code>

LXI Class B Triggering (IEEE-1588)

In this section:

Introduction to IEEE-1588 based triggering	11-1
IEEE-1588 implementation in the Series 3700.....	11-1
Correlating PTP to Coordinated Universal Time (UTC)...	11-2
Configuring and enabling IEEE-1588.....	11-3
Monitoring alarms with LAN triggers and LXI event log...	11-6

Introduction to IEEE-1588 based triggering

The Series 3700 uses IEEE-1588 Precision Time Protocol (PTP) to implement synchronized measurements and initiate time-triggered events over the LAN (Ethernet) interface. IEEE-1588 is a requirement of the LXI B Functional Class. Using IEEE-1588, you can schedule instrument-driven actions, such as measurements, to occur at a specific date and time and synchronize timebases between instruments on the same network. You can only access these capabilities through the remote command interfaces.

NOTE You can find detailed information on the syntax and usage of each ICL command presented in this section in *Instrument Control Library (ICL)* (on page 12-1).

IEEE-1588 implementation in the Series 3700

When you enable IEEE-1588 on a Series 3700 on a local network, the Series 3700 communicates with other IEEE-1588 enabled devices on the network through a dedicated network port called the PTP port. A predetermined algorithm then automatically selects the network device with the most accurate clock. This network device becomes the IEEE-1588 master. If multiple devices have the same clock accuracy, the protocol arbitrarily chooses one device to be the IEEE-1588 master.

When the protocol selects the Series 3700 as the master clock, the Series 3700 uses the time value stored in its battery-backed real-time clock and updates the time in all slave devices. When the protocol selects another networked device as the master clock, the Series 3700 is slave to the more accurate device and adjusts its time to that of the master clock. Additionally, the Series 3700 updates its battery-backed clock so that the time is 'remembered' if the master clock is removed from the network.

At periodic intervals, the master clock synchronizes to all slave clocks through time-stamped messages over the PTP port. This allows IEEE-1588 to maintain time synchronization between multiple devices on a network.

Program the synchronization interval in the Series 3700 using `ptp.syncinterval` (on page 13-229). The default synchronization interval is two seconds. Increasing the synchronization interval to values of more than two seconds increases the amount of time that it takes devices on the LAN to synchronize. If you change the synchronization interval, you must restart the clock of the Series 3700 by cycling its power.

Read the current time delay and offset between any slave device and its master on the LAN using `ptp.ds.current` (on page 13-223). Synchronization of time stamps between IEEE-1588 enabled devices to within 150ns can take as long as 2 minutes.

Correlating PTP to Coordinated Universal Time (UTC)

To ensure synchronization across networked devices, you must be aware of the time protocol utilized by those other devices on the network.

The most widely accepted time scale is Coordinated Universal Time (UTC); in many places, it is considered standard time. UTC is nearly the same time as Greenwich Mean Time (GMT), another very familiar time scale, and for the purposes of the Series 3700, UTC and GMT are the same. Local time is offset from UTC according to time zones; additional offsets can occur due to Daylight Savings Time adjustments.

UTC suffers from discontinuities because of non-periodic adjustments known as “leap seconds”. These adjustments present problems because they can make events that occurred at different periods of time appear to occur at the same time. PTP is a time standard that does not have any discontinuities and has no adjustments for local time (that is, it is not time-zone aware). PTP is presented as the number of seconds since January 1, 1970.

The Series 3700 offers two versions of time for most IEEE-1588-related commands, `.seconds` and `.ptpseconds`, representing UTC and PTP respectively. IEEE-1588 requires that devices are synchronized using UTC or PTP time, not local time. The Series 3700 does not distinguish UTC, PTP, and local time; it is not time-zone aware. You must be aware of this when synchronizing with devices that are time-zone aware.

When IEEE-1588 selects a time-zone aware device to be the master clock, the Series 3700 accepts the time of that clock. This time may not agree with the local time of the Series 3700, especially when a network spans multiple time zones. If you schedule events on the Series 3700 to occur according to your local time, events will not occur at the time you expect.

You can avoid confusion by setting the time on the Series 3700 to UTC time instead of local time. Manage the conversion from UTC to local time in your software application. For example, assume local time is Eastern Standard Time in the United States (EST), which is equivalent to GMT-5 (hours). Therefore, if the current local time is 3:00PM, the UTC time is 8:00PM. Set the time of the Series 3700 clock to 8:00PM. If it is then synchronized with a time-zone aware master clock, its time will not change significantly.

NOTE The Series 3700 does not differentiate UTC and PTP time. `ptp.utcoffset` (on page 13-229) is zero unless a master clock that is aware of the difference between UTC and PTP time populates this value. This value is volatile and not remembered through a power cycle.

Configuring and enabling IEEE-1588

To configure IEEE-1588, connect the Series 3700 to the LAN, along with any other IEEE-1588 enabled devices that you want to synchronize to the Series 3700. Refer to *Series 3700 Quick Start Guide* for information on connecting the Series 3700 to the LAN. If you want to synchronize multiple Series 3700 instruments on a LAN, each instrument must have the same PTP subdomain name.

The default PTP subdomain name is `_DFLT` for all Series 3700 devices. Use the `ptp.subdomain` function to change the subdomain name for any Series 3700 on the LAN. After changing the subdomain name, you must power cycle the Series 3700 to restart its clocks. If you have changed the subdomain name of any third-party IEEE-1588 enabled device within that subdomain, you must also restart its clock.

NOTE Cycling the power to the Series 3700 does not return the IEEE-1588-related parameters to factory default state. To return these to factory defaults, perform a LAN configuration reset. This can be done using `lan.status.reset()` (on page 13-202) on the remote command interface. You can also perform a reset through the front panel interface by entering the Main menu, selecting **LAN**, and selecting **Reset**.

Use `ptp.enable()` (on page 13-227) to enable IEEE-1588 on the Series 3700. The IEEE-1588 protocol then determines the master clock. The IEEE-1588 indicator on the front panel of the Series 3700 updates to display the IEEE-1588 status.

- If the indicator is off, then IEEE-1588 is disabled or the device is not connected to a working network.
 - If the network is not working, then the LAN indicator also blinks. If the indicator is solidly on, the IEEE-1588 is successfully enabled and synchronized, and the Series 3700 is a slave clock.
- If the network is not working, then the LAN indicator also blinks. If the indicator is solidly on, the IEEE-1588 is successfully enabled and synchronized, and the Series 3700 is a slave clock.
- If the indicator blinks once per second, then IEEE-1588 is successfully enabled and synchronized, and the Series 3700 is the master clock.
- If the indicator blinks once every two seconds, then IEEE-1588 is successfully enabled and synchronized, and the Series 3700 is the grandmaster clock.

You can also use `ptp.synchronized` (on page 13-228) to determine if the Series 3700 is a master or slave on the LAN.

NOTE `ptp.enable` is a non-volatile setting. Therefore, if you power off a Series 3700 with IEEE-1588 enabled and then re-power the Series 3700 on a different network, it attempts to synchronize with any other IEEE-1588 enabled devices on that new network. You do not need to re-enable IEEE-1588.

Scheduling alarms

You can schedule alarms to request the Series 3700 to perform actions at a specific time and date or at a specific time interval. You can schedule alarms in UTC or PTP time; however, it is important to be consistent in defining the alarms using the same time format whether UTC or PTP. Otherwise, the alarms will fire on the networked devices at different times, with a time difference equal to the PTP UTC offset.

You can set a maximum of two alarms per Series 3700.

To schedule an alarm, first convert the desired alarm time to UTC seconds. You can perform this conversion using `os.time`. If you are specifying alarms in UTC time, then you can use this value with `schedule.alarm[x].seconds` (on page 13-251) to schedule an alarm, where `x` represents the tag number of the alarm that you configure.

NOTE `os.time` is a LUA function that can be used to return the current time or convert a local date and time to UTC-based seconds elapsed since January 1, 1970. When used without parameters, `os.time` returns the current date and time. When used with parameters, the syntax is `os.time{year = <n>, month = <n>, day = <n>, hour = <n>, sec = <n>, isdst = }`. `<n>` is a number and `` is a Boolean where true is Daylight Savings Time. It is not necessary to specify all parameters.

The following examples demonstrate how to use `os.time`:

```
-- retrieve current UTC time in seconds since 1/1/1970
print(os.time)

-- convert 3:00PM March 1, 2008 to UTC seconds since
  1/1/1970:
local l_start_Time
l_start_Time = os.time{year=2008, month=3, day=1, hour=15}

-- create start time to occur 60 seconds after current time
local l_start_Time
l_start_Time = os.time() + 60
```

If you want to specify alarms in PTP format, convert UTC seconds to PTP seconds by adding the value returned by `ptp.utcoffset` (on page 13-229) to the UTC time. The Series 3700 alone does not differentiate PTP and UTC time. The `ptp.utcoffset` is only non-zero if the Series 3700 communicates to a master clock that is aware of the difference between PTP and UTC time. Use the converted PTP time in setting values for `schedule.alarm[x].ptpseconds` (on page 13-251), where `x` represents the tag number of the alarm you configure.

You can also schedule alarms to occur at a fractional second using either PTP or UTC format with `schedule.alarm[x].fractionalseconds` (on page 13-250).

After defining the alarm, configure the number of times you would like to repeat this alarm using `schedule.alarm[x].repetition` (on page 13-251). Use `schedule.alarm[x].period` (on page 13-251) to configure the amount of time, in seconds, between adjacent firings of the alarm. If you want the alarm to fire just once, set `schedule.alarm[x].period` to zero. If you want the alarm to repeat forever, set `schedule.alarm[x].period` to a non-zero value and set `schedule.alarm[x].repetition` to zero.

Enable the alarm by setting `schedule.alarm[x].enable` (on page 13-250) to 1. Disable an alarm by setting `schedule.alarm[x].enable` to 0. Disable all alarms using `schedule.disable()` (on page 13-252).

Monitoring alarms with LAN triggers and LXI event log

Use the LXI event log to monitor the firing of scheduled alarms. The LXI event log in the Series 3700 only captures LAN triggers that occur within its defined LXI domain. To monitor alarms, configure the alarm to generate a LAN trigger by using `schedule.alarm[x].EVENT_ID` (on page 13-250) as the control source for `lan.trigger[N].stimulus` (on page 13-208) in the trigger model. You can define up to eight LAN triggers.

Use `lan.lxidomain` (on page 13-197) to specify the LXI domain. Additionally, you can broadcast LAN triggers to all devices on a LXI domain, or you can transmit LAN triggers between two individual devices. To configure the LAN trigger broadcast, use `lan.trigger[N].protocol` (on page 13-207).

The following example demonstrates how to generate a LAN trigger when a scheduled alarm fires:

```
-- configure the LXI domain
lan.lxidomain=0

-- configure the LXI trigger to broadcast to all devices in
  this LXI domain
lan.trigger[2].protocol=2
lan.trigger[2].connect()

-- associate the firing of the alarm to the generation of a
  LAN trigger
lan.trigger[2].stimulus = schedule.alarm[1].EVENT_ID
```

LXI event log

The LXI event log of a Series 3700 monitors all LAN triggers that the instrument receives or generates. The LXI event log has nine comma-delimited fields. Below is an example entry to a LXI event log and a description of the log fields in order of appearance.

```
"17:26:35.690 10 Oct 2007, LAN0, 192.168.1.102, LXI, 0,
 1192037132, 1192037155.733269000, 0, 0x0"
```

Field #	Field Value	Field Description
1	"17:26:35.690 10 Oct 2007"	Formatted UTC time in 24-hour format including fractional seconds.
2	"LAN0"	Event identifier. NOTE This event identifier is zero-based (LAN0-LAN7). When specifying the LAN trigger using <code>lan.trigger[N]</code> , the minimum value for N is 1. Therefore LAN0 to LAN 7 corresponds to <code>lan.trigger[1]</code> through <code>lan.trigger[8]</code> , respectively.
3	"192.168.1.102"	IP address of the device that issued the LAN trigger.
4	"LXI"	LXI version identifier. Currently only LXI is defined.
5	"0"	LXI Domain number.
6	"1192037132"	Sequence number provided by the device that issued the LAN trigger. This number is incremented after generation of each LAN trigger.
7	"1192037155.733269000"	PTP time formatted as a floating point number.
8	"0"	The "overflow" from PTP seconds. Currently, this is "0". Also referred to as IEEE-1588 Epoch.
9	"0x0"	Hex value of the flag field, which is the logical OR of several conditions (error=1, retransmission=2, hardware=4, acknowledgement=8).

Example applications of IEEE-1588 in Series 3700-based systems

This section discusses examples of a few applications that are possible using IEEE-1588.

Scheduling alarms on a stand-alone Series 3700

To configure a single Series 3700 to perform an event at a particular date and time, you must schedule alarms, but you do not need to enable IEEE-1588. Therefore, you can send these commands over any remote interface and not just LAN.

To initiate a specific action at the firing of the alarm, you must use the event identifier for the scheduled alarm, `schedule.alarm[x].EVENT_ID` (on page 13-250), as the stimulus of one of the control sources defined in the trigger model. Refer to [Scanning](#) (on page 7-1) for more details on the trigger model.

The following example demonstrates how to configure a scan of five channels to run once every hour starting at 3AM on September 1, 2008:

```
-- convert to UTC time
Start_time = os.time({year=2008, month=9, day=1, hour=3})

-- convert to PTP time
Start_time = Start_time + ptp.utcoffset

-- configure the alarm
schedule.alarm[1].ptpseconds = Start_time
    schedule.alarm[1].fractionalseconds = 0

-- configure the alarm repetition count
schedule.alarm[1].repetition = 5

-- set alarm period to 1 hr = 60 secs x 60 mins
schedule.alarm[1].period = 60*60

--enable the alarm
schedule.alarm[1].enable = 1

--associate a DMM configuration and configure a scan
dmm.setconfig("1001:1005", "dcvolts")
scan.create("1001:1005")

    -- 5 scans of 5 channels
    buf = dmm.makebuffer(25)

--command the scan to start when alarm 1 fires
scan.trigger.arm.stimulus = schedule.alarm[1].EVENT_ID

--set scan count and initiate execution of background scan
scan.scancount = 5
scan.background(buf)
```

Synchronizing multiple Series 3700 instruments

NOTE Synchronization only occurs between instruments within the same PTP subdomain. Use `ptp.subdomain` to set the subdomain name. Refer to [Configuring and enabling IEEE-1588](#) (on page 11-3) for further details.

To execute synchronized actions on multiple Series 3700 instruments, you must connect these instruments to a network using Ethernet and enable IEEE-1588 protocol. Refer to [Configuring and enabling IEEE-1588](#) (on page 11-3) for details on how to enable IEEE-1588 protocol.

The protocol aligns the timebases of each of these instruments. You need to enter a delay in your software application to allow for this synchronization to complete. Synchronization to within 150ns can take as long as two minutes for system alignment. You can read the current time delay and offset between any slave device and its master on the LAN using `ptp.ds.current`. Refer to [IEEE-1588 Implementation in the Series 3700](#) (on page 11-1) for information on the time synchronization process.

After you enable IEEE-1588 and set up alarms, configure the action to occur at the firing of the alarm. Use the event identifiers for the scheduled alarms, `schedule.alarm[x].EVENT_ID` (on page 13-250), as the stimulus for the control sources in the trigger model.

Coordinating the Series 3700 with a device that is not IEEE-1588 enabled using scheduled alarms and digital I/O

If you have a network where the Series 3700 is the only IEEE-1588 enabled device, you can trigger actions on the other networked devices by means of digital triggers if such devices can process digital triggers. In this way, you can schedule alarms to execute switch-only or switch with DMM operations on the Series 3700 and you can also output triggers to other devices at the firing of the scheduled alarms.

The following is an example that demonstrates scheduling an alarm to execute a scan on the Series 3700 and output a trigger from the Series 3700 to another device:

```
-- configure a switch with DMM scan operation on a Series
  3700
reset()
scan.reset()
buffer=dmm.makebuffer(100)
dmm.autodelay=dmm.OFF
dmm.range=10
dmm.autozero=dmm.OFF
dmm.nplc=.0005
dmm.measurecount=1
dmm.configure.set('mydcvolts')
dmm.setconfig('1001:1010', 'mydcvolts')

scan.create('1001:1010')
scan.measurecount=1
scan.scancount = 10

-- configures falling-edge output trigger pulse as DIO line
  1 when generated, this hardware trigger would effect an
  action on the device that is not IEEE-1588 enabled
digio.trigger[1].mode = digio.TRIG_FALLING
digio.trigger[1].pulsewidth = 0.010
digio.trigger[1].clear()

-- execute scan to start when alarm fires
scan.trigger.arm.stimulus = schedule.alarm[1].EVENT_ID

-- generate output trigger when alarm fires
digio.trigger[1].stimulus = schedule.alarm[1].EVENT_ID

-- configure alarm to start 15 seconds after current PTP
  time
sec,ns=ptp.time()
schedule.alarm[1].ptpseconds=sec+15
schedule.alarm[1].fractionalseconds=0

-- configure alarm to fire 10 times at a period of 500ms
schedule.alarm[1].repetition=10
schedule.alarm[1].period=0.500

-- enable alarm
schedule.alarm[1].enable=1

-- initiate execution of foreground scan
scan.execute(buffer)
```

Status Model

In this section:

Status register sets	12-1
Status byte and SRQ	12-2
System summary and status byte	12-3
System summary registers.....	12-4
Standard event status register and enable.....	12-5
Operation events registers	12-6
Questionable event register	12-7
Measurement event register (measurement)	12-8
Status function summary.....	12-8
Clearing registers and queues	12-9
Programming enable and transition registers.....	12-10
Reading registers	12-11
Status byte and service request (SRQ).....	12-12
Status register set specifics	12-16
Queues	12-25

The Keithley Instruments Series 3700 System Switch/Multimeter provides a number of status registers and queues that allow the operator to monitor and manipulate various instrument events. 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 the SRQ.

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.

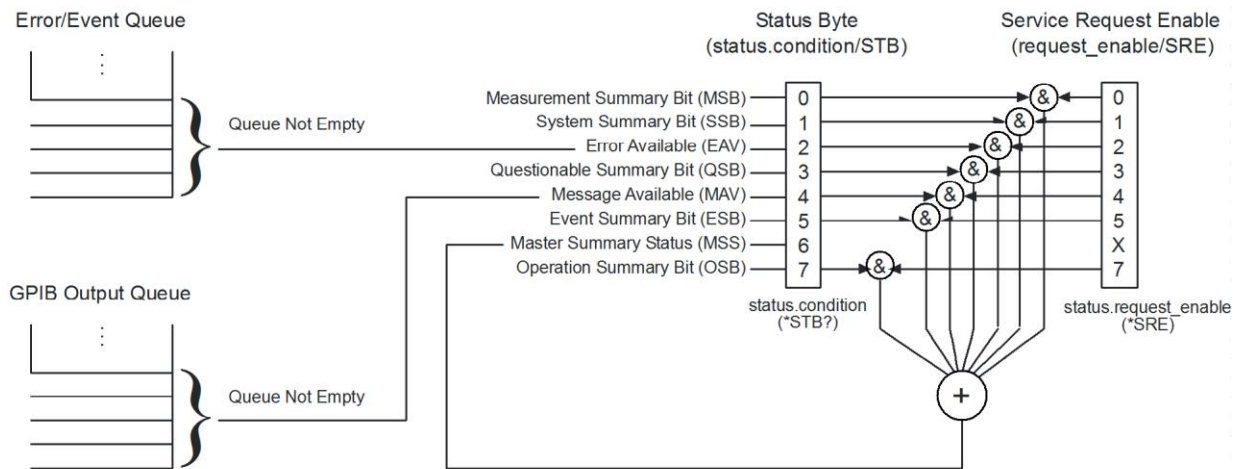
Negative and positive transition registers

- **Negative-transition register (NTR):** When a bit in an NTR 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 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, a master summary bit, 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.

Figure 12-1: Status byte and queues



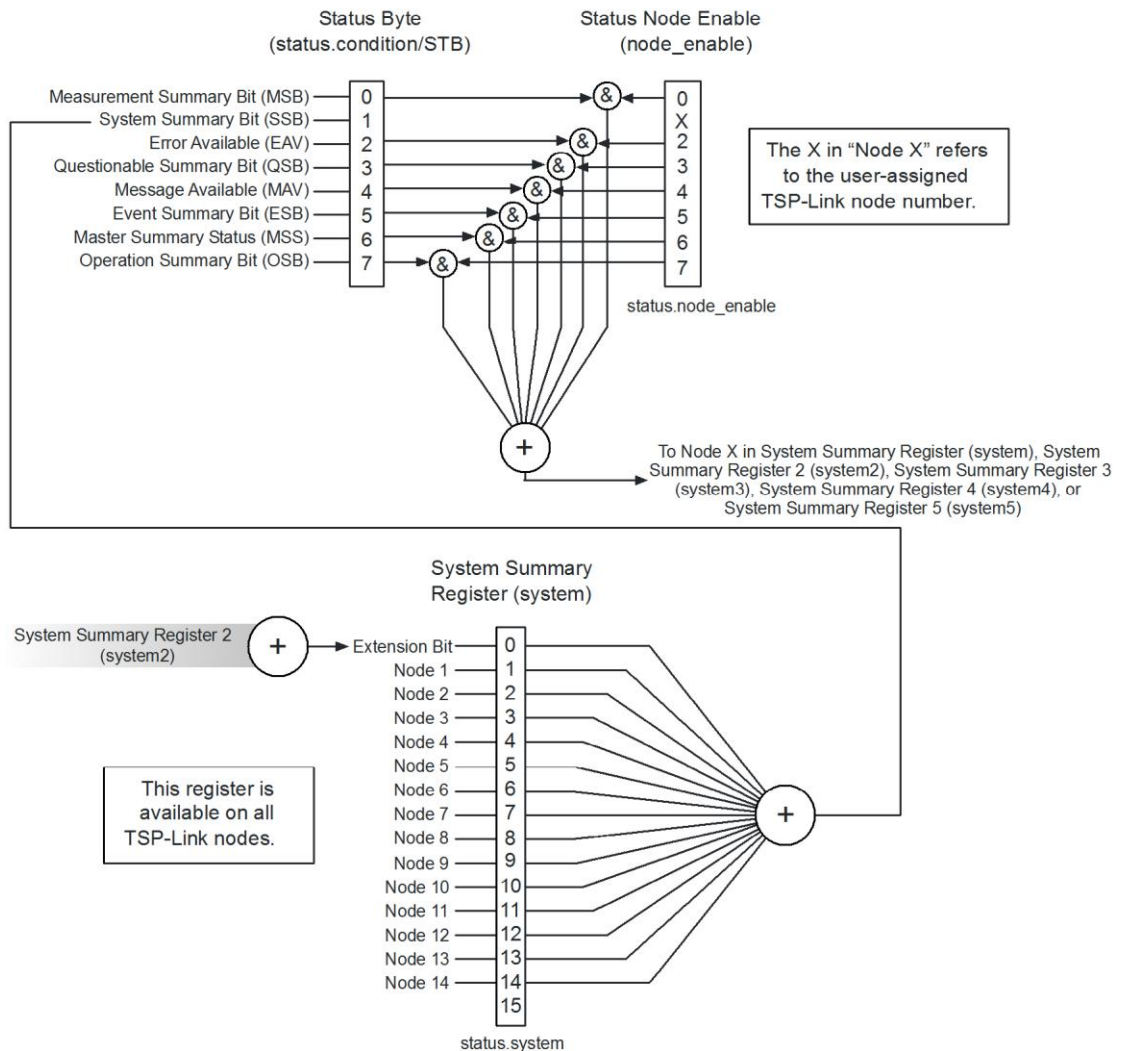
Queues

The Series 3700 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).

System summary and status byte

The system summary bit (SSB) is received by the status byte (Bit 1) from the system summary register (`status.system`) byte. The summary of system summary register (`status.system`) receives its extension bit (Bit 0) from the system summary register 2 (`status.system2`).

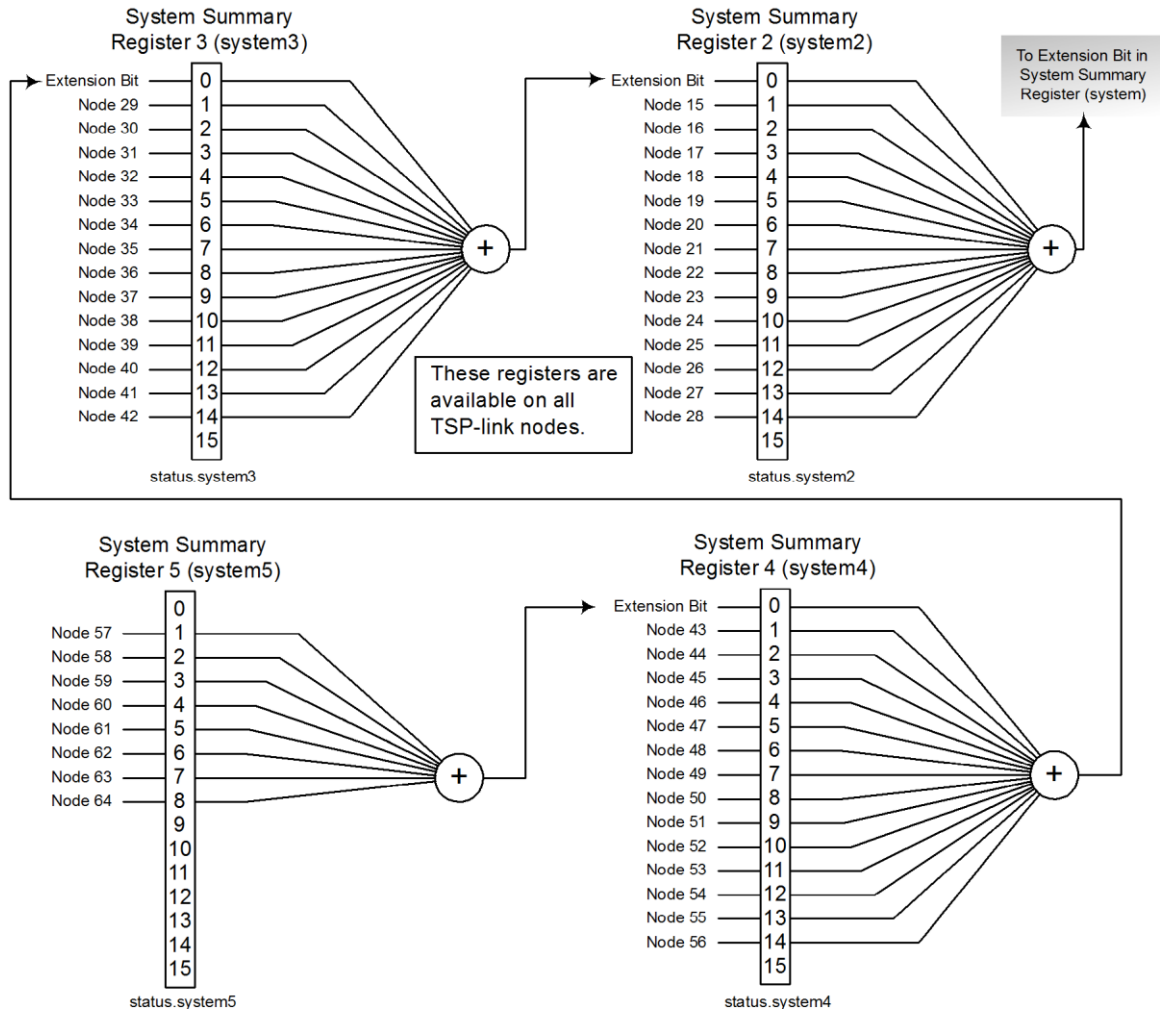
Figure 12-2: Status byte and system summary register



System summary registers

The system summary registers (system5 through system2) provide summary information through the specific register's extension bit (Bit 0). This in turn is provided to the system summary register (system) extension bit (Bit 0).

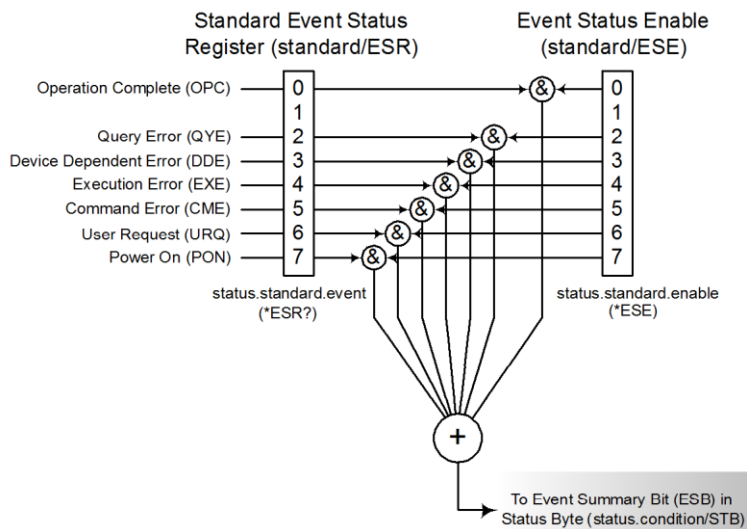
Figure 12-3: System summary registers



Standard event status register and enable

The summary bit of the standard event status register and event status enable provide summary information to Bit 5 of the status byte (*Status byte and SRQ* (on page 12-2)).

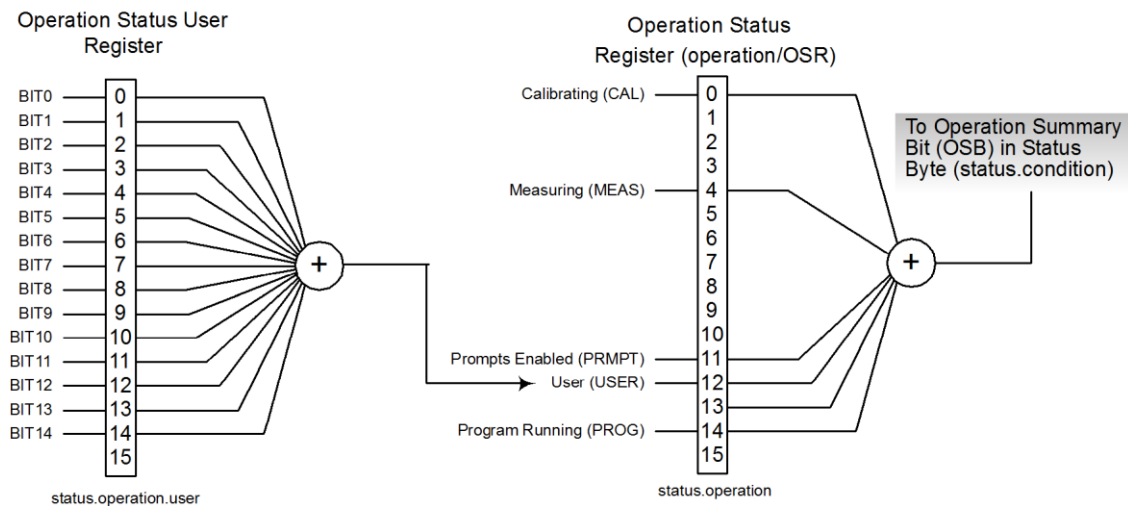
Figure 12-4: Standard event registers and event status enable



Operation events registers

The summary bit of the operation status user register provides the user bit (USER) (Bit 12) to the operation status register. The summary bit of the operation status register provides the operation summary bit (OSB) (Bit 7) of the status byte.

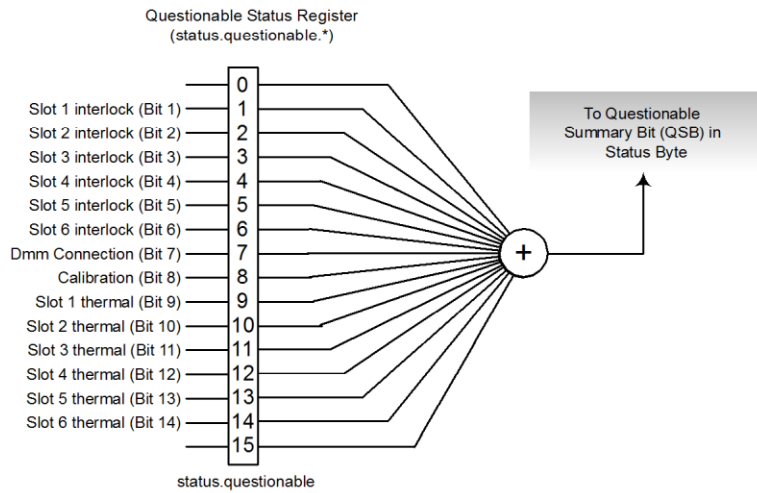
Figure 12-5: Operation event registers



Questionable event register

The questionable event register provides summary information to questionable summary bit (QSB) (Bit 3) of the status byte.

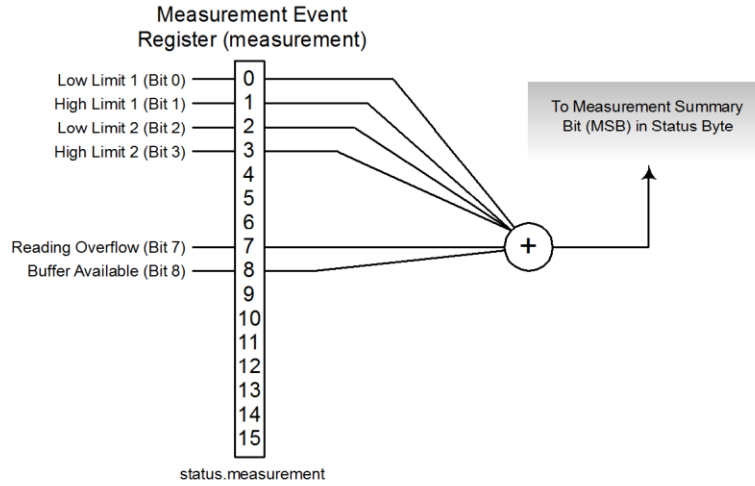
Figure 12-6: Questionable event register



Measurement event register (measurement)

The measurement event register (measurement) provides summary information to the status byte's measurement summary bit (MSB) (Bit 0).

Figure 12-7: Measurement event register



Status function summary

The following functions control and read the various registers.

NOTE * = .ntr, .ptr, .enable, .event, and .condition. The first three (.ntr, .ptr, and .enable) are read/write, while the last two (.event and .condition) are read only.

Type	Function
System summary	status.reset status.node_enable status.request_enable status.node_event status.request_event status.condition
Measurement event	status.measurement.*
Operation event	status.operation.* status.operation.user.*

Type	Function
Questionable event	<code>status.questionable.*</code>
Standard event	<code>status.standard.enable</code>
System event	<code>status.system.enable</code> <code>status.system2.enable</code> <code>status.system3.enable</code> <code>status.system4.enable</code> <code>status.system5.enable</code>

Clearing registers and queues

When the Series 3700 is turned on, various register status elements will be set as follows:

- The power on (PON) bit in the `status.operation.condition` register is set.
- All enable registers are set to 0.
- All negative-transition registers (NTRs) are set to 0.
- All used positive transition register (PTR) bits are set to 1.
- The two queues are empty.

Commands to reset the status registers and the error queue are listed in the following table. 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.

To reset registers:

```
-- Reset bits of status registers to 0.
status.reset()
```

To clear the error queue:

```
-- Clear all messages from the error queue.
errorqueue.clear()
```

Programming enable and transition registers

The only registers that can be programmed by the user are the enable and transition registers. All other registers in the status structure are read-only registers.

To determine the parameter values for the various commands used to program enable registers, 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 (see the following figure) 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.

Figure 12-8: 16-bit status register

Bit Position	B7	B6	B5	B4	B3	B2	B1	B0
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 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	32768	16384	8192	4096	2048	1024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(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 the figure above.

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 by either 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, the following commands request the Service Request Enable register value:

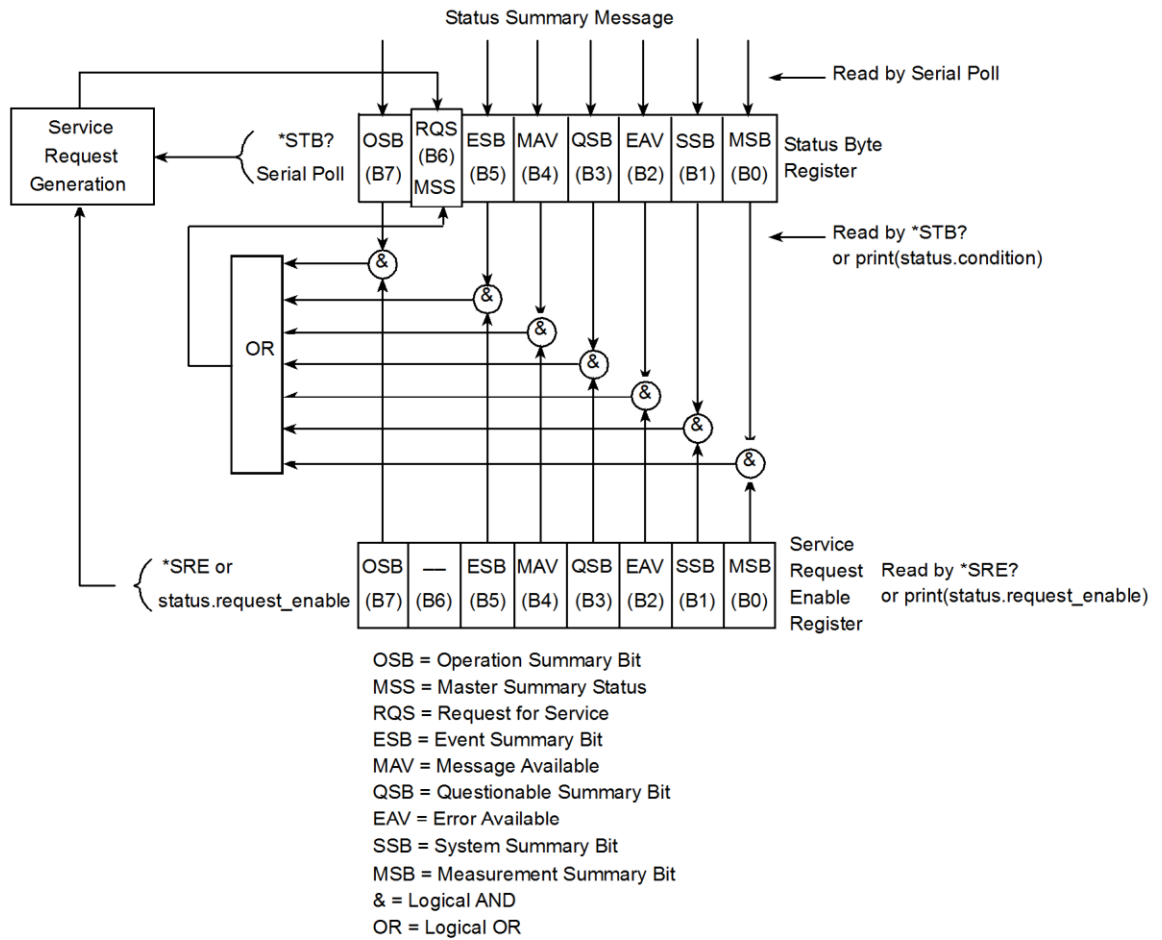
```
*SRE?  
print(tostring(status.request_enable))  
print(status.request_enable)
```

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. If the response is 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 12-9: Status byte and service request (SRQ)



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 event summary bit (ESB) bit in the status byte register.

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 Series 3700 to obtain the status byte (serial poll byte), B6 is the RQS bit. See [Serial polling and SRQ](#) (on page 12-14) for details about using the serial poll sequence.

When using the `*STB?` common command or `status.condition` to read the status byte, B6 is the MSS bit.

- **Bit B7, operation summary bit (OSB):** Set summary bit indicates that an enabled operation event has occurred.

Serial polling and SRQ

Any enabled event summary bit that goes from 0 to 1 sets bit B6 and generates 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 Series 3700. 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.

Service request enable register

The generation of a service request is controlled by the Service Request Enable Register. You program this register and use it 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. 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 (for example, *SRE 0).

SPE, SPD (serial polling)

For the GPIB interface only, the SPE and SPD general bus command sequence is used to serial poll the Series 3700. 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 below. Note that the table includes both common commands and their script command equivalents. For details on programming and reading registers, see [Programming enable and transition registers](#) (on page 12-10) and [Reading registers](#) (on page 12-11).

To reset the bits of the service request enable register to 0, use 0 as the parameter value for the command (for example, *SRE 0).

Commands	Description
*STB? or <code>print(status.condition)</code>	Read status byte register.
*SRE <mask> or <code>status.request_enable = <mask></code>	Program the service request enable register: <mask> = 0 to 255
*SRE? or <code>print(status.request_enable)</code>	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 register summary 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

For the attributes to control system node and SRQ enable bits and read associated registers, refer to:

- `status.request_enable` (on page 13-276)
- `status.request_event` (on page 13-278)
- `status.node_enable` (on page 13-267)
- `status.node_event` (on page 13-269)

Status register set specifics

There are five status register sets in the status structure of the Series 3700:

- System summary event status
- Standard event status
- Operation event status
- Questionable event status
- Measurement event status

System summary event registers

There are five register sets associated with system event status. These registers summarize system status for various nodes connected to the TSP-Link™. Note that all nodes on the TSP-Link share a copy of the system summary registers once the TSP-Link has been initialized. This feature allows all nodes to access the status models of other nodes, including SRQ. Commands are summarized below.

For example, either of the following commands will set the extension (EXT) enable bit:

```
status.system.enable = status.system.EXT
status.system.enable = 1
```

The following command will request the system enable register value:

```
print(status.system.enable)
```


The used bits of the system event registers are described as follows:

- **EXT:** Set bit indicates that an extension bit from a another system status register is set.
- **NODEn:** Indicates a bit on TSP-Link node n has been set (n = 1 to 64).

System summary event commands appear in the following table:

Commands	Bit
To set register bits: <pre>status.system.enable = status.system.EXTENSION_BIT status.system.enable = status.system.EXT status.system.enable = status.system.NODEn</pre>	B0 B0 Bn
To read registers: <pre>print(status.system.enable) print(status.system.condition) print(status.system.event)</pre>	
To set register bits: <pre>status.system2.enable = status.system.EXTENSION_BIT status.system2.enable = status.system.EXT status.system2.enable = status.system.NODEn</pre>	B0 B0 Bn
To read registers: <pre>print(status.system2.enable) print(status.system2.condition) print(status.system2.event)</pre>	
To set register bits: <pre>status.system3.enable = status.system.EXTENSION_BIT status.system3.enable = status.system.EXT status.system3.enable = status.system.NODEn</pre>	B0 B0 Bn
To read registers: <pre>print(status.system3.enable) print(status.system3.condition) print(status.system3.event)</pre>	
To set register bits: <pre>status.system4.enable = status.system.EXTENSION_BIT status.system4.enable = status.system.EXT status.system4.enable = status.system.NODEn</pre>	B0 B0 Bn

Commands	Bit
To read registers: <pre>print(status.system4.enable) print(status.system4.condition) print(status.system4.event)</pre>	
To set register bits: <pre>status.system5.enable = status.system.EXTENSION_BIT status.system5.enable = status.system.EXT status.system5.enable = status.system.NODEn</pre>	B0 B0 Bn
To read registers: <pre>print(status.system5.enable) print(status.system5.condition) print(status.system5.event)</pre>	

Refer to the following table for available n values:

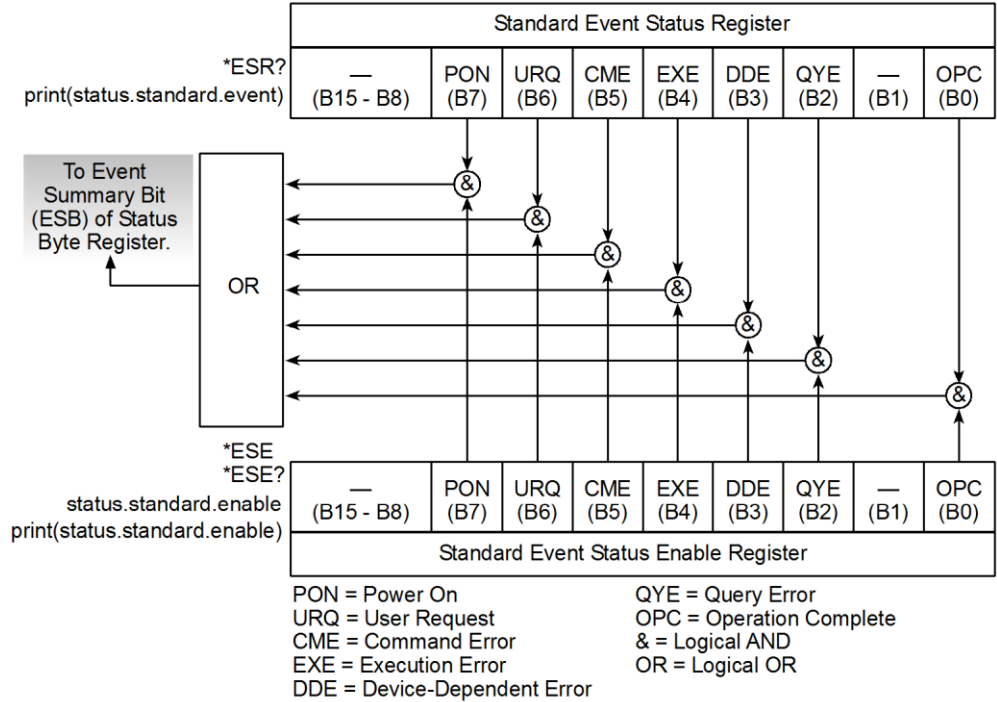
Command	n value
status.system	1 to 14
status.system2	15 to 28
status.system3	29 to 42
status.system4	43 to 56
status.system5	57 to 64

Standard event register

The bits used in the standard event register are described as follows:

- **Bit B0, operation complete:** Set bit indicates that all pending selected device operations are completed and the Series 3700 is ready to accept new commands. The bit is set in response to an *OPC command. The ICL function `opc()` (on page 13-220) can be used in place of the *OPC command.
- **Bit B1:** Not used.
- **Bit B2, query error (QYE):** Set bit indicates that you attempted to read data from an empty output queue.
- **Bit B3, device-dependent error (DDE):** Set bit indicates that an instrument operation did not execute properly due to some internal condition.
- **Bit B4, execution error (EXE):** Set bit indicates that the Series 3700 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:** Series 3700 received a message that does not follow the defined syntax of the IEEE-488.2 standard.
 - **Semantic error:** Series 3700 received a command that was misspelled or received an optional IEEE-488.2 command that is not supported.
 - **Group execute trigger location error:** 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 Series 3700 front panel was pressed.
- **Bit B7, Power ON (PON):** Set bit indicates that the Series 3700 has been turned off and turned back on since the last time this register was read.

Figure 12-10: Standard event register



Commands to program and read the register are summarized in the table below. Refer to the `status.standard` ICL command for the attributes to control bits.

Commands	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 Refer to the <code>status.standard.</code> (on page 13-278) ICL command
*ESE? or print(status.standard.enable)	Read event status enable register.

Operation event registers

The Series 3700 has two register sets associated with operation event status. Commands are summarized in the following table. For example, either of the following commands will set the CAL enable bit:

```
status.operation.enable = status.operation.CAL
```

```
status.operation.enable = 1
```

The bits used in the operation event registers are described as follows:

- **CAL:** Set bit indicates that the instrument is calibrating.
- **MEAS:** Bit will be set when taking an overlapped measurement, but it will not set when taking a normal synchronous measurement.
- **PRMPT:** Set bit indicates that command prompts are enabled.
- **USER:** Set bit indicates that an enabled bit in the operation status user register is set.
- **PROG:** Set bit indicates that a program is running.

Operation event commands appear in the following table:

Commands	Bit
To set register bits: <pre>status.operation.* = status.operation.CAL or status.operation.CALIBRATING</pre> <pre>status.operation.* = status.operation.MEAS or status.operation.MEASURING</pre> <pre>status.operation.* = status.operation.PRMT or status.operation.PROMPTS</pre> <pre>status.operation.* = status.operation.USER</pre> <pre>status.operation.* = status.operation.PROG or status.operation.PROGRAM_RUNNING</pre> To read registers: <pre>print(status.operation.*)</pre> Where * = .ntr, .ptr, .enable, .event, and .condition	B0 B4 B11 B12 B14
NOTE The first three (.ntr, .ptr, and .enable) are read/write, while the last two (.event and .condition) are read only. <pre>print(status.operation.condition)</pre> <pre>print(status.operation.event)</pre>	
To set register bits: <pre>status.operation.user.* = status.operation.user.BIT0</pre>	B0

Commands	Bit
<code>status.operation.user.* = status.operation.user.BIT1</code>	B1
<code>status.operation.user.* = status.operation.user.BIT2</code>	B2
<code>status.operation.user.* = status.operation.user.BIT3</code>	B3
<code>status.operation.user.* = status.operation.user.BIT4</code>	B4
<code>status.operation.user.* = status.operation.user.BIT5</code>	B5
<code>status.operation.user.* = status.operation.user.BIT6</code>	B6
<code>status.operation.user.* = status.operation.user.BIT7</code>	B7
<code>status.operation.user.* = status.operation.user.BIT8</code>	B8
<code>status.operation.user.* = status.operation.user.BIT9</code>	B9
<code>status.operation.user.* = status.operation.user.BIT10</code>	B10
<code>status.operation.user.* = status.operation.user.BIT11</code>	B11
<code>status.operation.user.* = status.operation.user.BIT12</code>	B12
<code>status.operation.user.* = status.operation.user.BIT13</code>	B13
<code>status.operation.user.* = status.operation.user.BIT14</code>	B14
To read registers:	
<code>print(status.operation.user.*)</code>	
Where * = .ntr, .ptr, .enable, .event, and .condition	
NOTE The first three (.ntr, .ptr, and .enable) are read/write, while the last two (.event and .condition) are read only.	
<code>print(status.operation.user.condition)</code>	
<code>print(status.operation.user.event)</code>	

Questionable event registers

The Series 3700 has five registers associated with questionable event status. For example, to set the CAL enable bit, use either of the following commands:

```
status.questionable.enable = status.questionable.CAL
status.questionable.enable = 256
```

The following command will request the questionable enable register value:

```
print(status.questionable.enable)
```

The bits used in the questionable event registers are described as follows:

- **SxINL**: Set bit indicates the interlock connection of a card in slot x is in question, where x = 1 to 6.
- **DMMCON**: Set bit indicates the DMM connection is in question for a measurement taken.
- **CAL**: Set bit indicates the calibration of the instrument is in question.
- **SxTHR**: Set bit indicates the thermal aspect of the card in slot x is in question, where x = 1 to 6.

Questionable event commands appear in the following table:

Commands	Bit
To set register bits:	
SLOT1_INTERLOCK or S1INL	B1
SLOT2_INTERLOCK or S2INL	B2
SLOT3_INTERLOCK or S3INL	B3
SLOT4_INTERLOCK or S4IN	B4
SLOT5_INTERLOCK or S5INL	B5
SLOT6_INTERLOCK or S6INL	B6
DMM_CONNECTION or DMMCON	B7
DMM_CALIBRATION or CAL	B8
SLOT1_THERMAL or S1THR	B9
SLOT2_THERMAL or S2THR	B10
SLOT3_THERMAL or S3THR	B11
SLOT4_THERMAL or S4THR	B12
SLOT5_THERMAL or S5THR	B13
SLOT6_THERMAL or S6THR	B14

Commands	Bit
To read registers: <pre>print(status.questionable.user.*)</pre> Where * = .ntr, .ptr, .enable, .event, and .condition	
NOTE The first three (.ntr, .ptr, and .enable) are read/write, while the last two (.event and .condition) are read only.	
<pre>print(status.questionable.user.condition) print(status.questionable.user.event)</pre>	

Measurement event registers

The Series 3700 has five registers associated with measurement event status. For example, to set the buffer available bit, use either of the following commands:

```
status.measurement.enable = status.measurement.BAV
status.measurement.enable = 256
```

The bits used in the measurement event registers are described as follows:

- **ROF:** Set bit indicates that an overflow reading has been detected.
- **BAV:** Set bit indicates that there is at least one reading stored in a reading buffer.
- **ULMT1:** Set bit indicates that a reading has exceeded the upper limit 1 value.
- **LLMT1:** Set bit indicates that a reading has exceeded the lower limit 1 value.
- **ULMT2:** Set bit indicates that a reading has exceeded the upper limit 2 value.
- **LLMT2:** Set bit indicates that a reading has exceeded the lower limit 2 value.

Measurement event commands appear in the following table:

Commands	Bit
To set register bits:	
<pre>LLMT1 or LOWER_LIMIT1</pre>	B0
<pre>ULMT1 or UPPER_LIMIT1</pre>	B1
<pre>LLMT2 or LOWER_LIMIT2</pre>	B2
<pre>ULMT2 or UPPER_LIMIT2</pre>	B3
<pre>ROF or READING_OVERFLOW</pre>	B7
<pre>BAV or BUFFER_AVAILABLE</pre>	B8

Commands	Bit
To read registers: <pre>print(status.measurement.*)</pre> Where * = .ntr, .ptr, .enable, .event, and .condition	
NOTE The first three (.ntr, .ptr, and .enable) are read/write, while the last two (.event and .condition) are read only.	
<pre>print(status.measurement.condition) print(status.measurement.event)</pre>	

Queues

The Series 3700 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.

The Series 3700 status model 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 and 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 Series 3700 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 and 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 the following table. 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" are placed in the queue. Messages in the error queue include a code number, message text, severity, and TSP-Link™ node number.

Error queue command	Description
<code>errorqueue.clear()</code>	Clear error queue of all errors.
<code>errorqueue.count</code>	Number of messages in the error/event queue.
<code>errorcode, message, severity, node = errorqueue.next()</code>	Request error code, text message, severity, and TSP-Link node number.

Instrument Control Library (ICL)

In this section:

Command programming notes.....	13-1
ICL command list	13-11

Command programming notes

Wild characters

For the following command reference, it is necessary to understand the following conventions. Many commands are expressed in a generic form using wild characters. A wild character indicates a channel, function, or trigger line. Remember 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 slot (1 through 6) and **Y** is used to indicate the limit number (1 or 2). For example, the attribute for the limit 2 number testing is generically expressed as follows:

```
dmm.limit[Y].enable
```

To enable limit 2, send the following command statement to the instrument:

```
dmm.limit[2].enable = dmm.ON
```

To query for idn information for the card in Slot 1, send the following command statement to the instrument:

```
card1_idn = slot[1].idn
```

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 3700 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](#) (on page 13-2)).

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). A function based command is not always directly related to a Series 3700 operation. For example, the `bit.bitand` function will logically AND two numbers.

Each function consists of a function name followed by a set of parenthesis (). If the function does not have a parameter, the parenthesis set is left empty.

Examples:

<code>digio.writeport(15)</code> high.	' Sets digital I/O lines 1, 2, 3, and 4
<code>digio.writebit(3, 0)</code>	' Sets line 3 low (0).
<code>dmm.reset('all')</code> settings.	' Returns the DMM to its default
<code>digio.readport()</code>	' Reads the digital I/O port.

The results of a function call are used by assigning the return values to variables and accessing those variables. The following code will measure voltage and return the reading:

```
dmm.func = 'dcvolts'  
  
reading = dmm.measure()  
  
print(reading)
```

Output: 2.360000e+00

The above output indicates that the voltage reading is 2.36V.

Attributes

An attribute is a characteristic of an instrument feature or operation. For example, some characteristics of a digital multimeter (DMM) include the measurement function and range.

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, `dmm.autorange` is an attribute. The autorange attribute is enabled by assigning the attribute to either of the following values:

1 or `dmm.ON`

Either of the following command messages will configure the DMM for the moving average filter:

```
dmm.filter.type = 0  
dmm.filter.type = dmm.FILTER_MOVING_AVG
```

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 the DMM by passing the attribute to the print function, which outputs a value:

```
print(dmm.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 = dmm.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:

```
dmm.func = dmm.DC_VOLTS
```

In the above command, which selects the DC volts measurement function, `dmm.DC_VOLTS` is the attribute constant and `dmm.func` is the attribute command.

- White space in a function is not required. The function to set digital I/O line 3 low can be sent with or without white spaces 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.

In relation to Test Script Processor (TSP™), 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](#) (on page 13-5)).

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. In the context of TSP™ and instrument control libraries, it is useful to think of individual DMMs as instruments. To avoid confusion, DMMs 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 3700 has the following logical instruments per enclosure:

- beeper
- bit
- channel
- digio
- display
- dmm
- eventlog
- errorqueue
- format
- gpib
- lan
- scan
- slot
- status
- timer
- trigger
- tsplink

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, `dmm`.

node: A reference to the TSP-Link™ node of which the logical instrument is a part. Default value is 2.

Query commands

Channel response query commands can return a comma-delimited string (for example, `channel.getcount` and other channel response query commands).

When a channel query command like `channel.getcount` or `channel.getstate` is sent, the response is a comma-delimited list. The list starts with the lowest channel through to the highest. After the channels are listed, the analog backplane relays are listed, starting with Bank 1 followed by each subsequent bank.

For example (Model 3720 card installed in Slot 4, returning 72 comma-delimited values):

Send the following command:

```
print(channel.getclose('slot4'))
```

The first 60 values returned are for Channels 1 to 60, starting with 1 and increasing to 60. The next six values are for analog backplane relays in Bank 1 (starting at 1 and increasing to 6). The final 6 values are for analog backplane relays in Bank 2 (again, starting at 1 and increasing to 6).

If the command was `channel.getstate` instead of `channel.getcount`, then 72 zero (0) or one (1) values would be returned (a 0 would indicate that the channel or backplane is open; a 1 would indicate that it is closed). The first 60 values are for Channels 1 to 60 (starting at 1 and increasing to 60). The last 12 values are the backplane relays (starting with Bank 1, Relay 1, increasing to Bank 2, Relay 6).

NOTE If a channel is paired for 4-wire by its pole setting, then the paired channel state is returned in parenthesis () after the primary channel. For example, if the card in Slot 4 is a Model 3720 and has the 4-pole attribute for all channels set, querying for the states of "slot4" will return 72 zeros and ones, with the first 60 shown as the primary channel state (paired channel state); the 12 backplane relays follow.

Sample code and output:

```
channel.setpole('slot4', 4)

print(channel.getstate('slot4'))
```

Output from above code:

```
0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0)
, 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0),
0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0 (0), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0
```

The Model 3721 card has three additional backplane relays for common side ohms functionality. Use 'slotX' or 'allslots' to query settings on this card to return information for Channels 1 to 40, 911 to 916, 921 to 926, and then 917, 927, and 928 in the response message (the three additional common side ohms backplane relays are listed last).

For example, to print out the channel images on this card when it is in Slot 2 after a reset, send the following:

```
reset()
print(channel.getimage('slot2'))
```

Output from above code:

```
2001;2002;2003;2004;2005;2006;2007;2008;2009;2010;2011;2012
;2013;2014;2015;2016;2017;2018;2019;2020;2021;2022;2023;202
4;2025;2026;2027;2028;2029;2030;2031;2032;2033;2034;2035;20
36;2037;2038;2039;2040;2041;2042;2911;2912;2913;2914;2915;2
916;2921;2922;2923;2924;2925;2926;2917;2927;2928
```

NOTE The common side ohm backplane relays (2917, 2927, and 2928) are listed last.

DMM configuration

When a DMM configuration is updated, the instrument will go through all switch channels in the system to find ones that have the updated configuration assigned as its DMM configuration attribute setting. For each one found, it will verify that the new configuration attribute settings are still valid for that channel. If the settings are still valid and needed, changes will be made to support the new settings.

The Model 3721 card has three additional backplane relays for common side ohms functionality. Using 'slotX' or 'allslots' to query settings on this card returns information for Channels 1 to 40, 911 to 916, 921 to 926, then 917, 927, and 928 in the response message (the three additional common side ohms backplane relays are listed last).

For example, to print the channel images on this card when it is in Slot 2 after a reset, send the following:

```
reset ()

print(channel.getimage('slot2'))
```

Output from above code:

```
2001;2002;2003;2004;2005;2006;2007;2008;2009;2010;2011;2012;2013;2014;2015;2016;2017;2018;2019;2020;2021;2022;2023;2024;2025;2026;2027;2028;2029;2030;2031;2032;2033;2034;2035;2036;2037;2038;2039;2040;2041;2042;2911;2912;2913;2914;2915;2916;2921;2922;2923;2924;2925;2926;2917;2927;2928
```

NOTE The common side ohm backplane relays (2917, 2927, and 2928) are listed last.

DMM new configuration example

For example, assume Slot 6 has a Model 3720 card installed and the following ICL commands are sent:

```
reset()
dmm.nplc = .1
dmm.range = 10
dmm.configure.set('myconfig')
dmm.setconfig('slot6', 'myconfig')
print(dmm.getconfig('slot6'))
```

The output will be 60 comma-delimited `myconfig` strings, as follows:

```
myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,mycon
fig,myconfig,myconfig,myconfig,myconfig,myconfig,myconfi
g,myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,
myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,myco
nfig,myconfig,myconfig,myconfig,myconfig,myconfig,myconfi
g,myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,
myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,m
yconfig,myconfig,myconfig,myconfig,myconfig,myconfig,myc
onfig,myconfig,myconfig,myconfig
```

Next, the following ICL commands are sent:

```
dmm.func = 'fourwireohms'
dmm.nplc = .5
dmm.range = 100000
dmm.configure.set('myconfig')
print(dmm.getconfig('slot6'))
```

The output will be 30 comma-delimited `myconfig` strings, as follows:

```
myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,mycon
fig,myconfig,myconfig,myconfig,myconfig,myconfig,myconfi
g,myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,
myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,my
config,myconfig,myconfig,myconfig,myconfig,myconfig
```

As this example shows, 'myconfig' was first saved with a function setting of DC volts, which was valid for all 60 channels on Slot 6. However, when 'myconfig' was associated with a function setting of 4-wire ohms, Channels 31 to 60 became unavailable because they are paired with Channels 1 to 30 in 4-wire measurement operation.

Next, the following ICL commands are sent:

```
dmm.func = 'temperature'
dmm.configure.set('myconfig')
print(dmm.getconfig('slot6'))
```

The output will be 30 comma-delimited `myconfig` strings, followed by 30 comma-delimited `nofunctions` strings, as follows:

```
myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,mycon
fig,myconfig,myconfig,myconfig,myconfig,myconfig,myconfi
g,myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,
myconfig,myconfig,myconfig,myconfig,myconfig,myconfig,my
config,myconfig,myconfig,myconfig,myconfig,nofunction,no
function,nofunction,nofunction,nofunction,nofunction,nof
unction,nofunction,nofunction,nofunction,nofunction,nofu
nction,nofunction,nofunction,nofunction,nofunction,nofun
ction,nofunction,nofunction,nofunction,nofunction,nofunct
ion,nofunction,nofunction,nofunction
```

If you now associate 'myconfig' with a 2-wire temperature function, 'myconfig' stays on Channels 1 to 30, but because Channels 31 to 60 are no longer paired, they are set back to a default DMM configuration setting of 'nofunction' because the unit has no way of knowing what configuration is desired on those channels.

Likewise, changing a DMM configuration to pertain to a function not supported by a channel will cause the channel to be set back to the 'nofunction' setting. Continuing the example, if 'myconfig' is set to pertain to the DC current function, then Channels 1 to 30, Slot 6, would be set to 'nofunction' because these channels don't support the amp functionality.

Next, the following ICL commands are sent:

```
dmm.func = 'dcurrent'  
dmm.configure.set('myconfig')  
print(dmm.getconfig('slot6'))
```

The output will be 60 comma-delimited nofunctions, as follows:

```
nofunction,nofunction,nofunction,nofunction,nofunction,nofu  
nction,nofunction,nofunction,nofunction,nofunction,nofun  
ction,nofunction,nofunction,nofunction,nofunction,nofunc  
tion,nofunction,nofunction,nofunction,nofunction,nofuncti  
on,nofunction,nofunction,nofunction,nofunction,nofunctio  
n,nofunction,nofunction,nofunction,nofunction,nofunction  
,nofunction,nofunction,nofunction,nofunction,nofunction,  
nofunction,nofunction,nofunction,nofunction,nofunction,n  
ofunction,nofunction,nofunction,nofunction,nofunction,no  
function,nofunction,nofunction,nofunction,nofunction,nof  
unction,nofunction,nofunction,nofunction
```

As this example demonstrates, it is important to be careful when updating the attributes associated with a DMM configuration. Changing settings within the same function (for example, NPLC or range) will usually not change the DMM configuration, but you should still be cautious. For example, you should be careful changing a setting like transducer for temperature to indicate 2-pole or 4-pole measurement operation.

ICL command list

This section provides a listing of Instrument Control Library commands and basic functional group usage.

beeper functions and attributes

[beeper.beep\(\)](#) (on page 13-16)
[beeper.enable](#) (on page 13-16)

bit functions

[bit.bitand\(\)](#) (on page 13-17)
[bit.bitor\(\)](#) (on page 13-18)
[bit.bitxor\(\)](#) (on page 13-18)
[bit.clear\(\)](#) (on page 13-19)
[bit.get\(\)](#) (on page 13-19)
[bit.getfield\(\)](#) (on page 13-20)
[bit.set\(\)](#) (on page 13-21)
[bit.setfield\(\)](#) (on page 13-21)
[bit.test\(\)](#) (on page 13-22)
[bit.toggle\(\)](#) (on page 13-23)

channel functions and attributes

[channel.clearforbidden\(\)](#) (on page 13-36)
[channel.close\(\)](#) (on page 13-37)
[channel.connectrule](#) (on page 13-39)
[channel.connectsequential](#) (on page 13-40)
[channel.exclusiveclose\(\)](#) (on page 13-41)
[channel.exclusiveslotclose\(\)](#) (on page 13-43)
[channel.getbackplane\(\)](#) (on page 13-45)
[channel.getclose\(\)](#) (on page 13-46)
[channel.getcount\(\)](#) (on page 13-48)
[channel.getdelay\(\)](#) (on page 13-49)
[channel.getforbidden\(\)](#) (on page 13-50)
[channel.getimage\(\)](#) (on page 13-51)
[channel.getlabel\(\)](#) (on page 13-52)
[channel.getpole\(\)](#) (on page 13-55)
[channel.getstate\(\)](#) (on page 13-56)
[channel.open\(\)](#) (on page 13-59)
[channel.pattern.catalog\(\)](#) (on page 13-61)
[channel.pattern.delete\(\)](#) (on page 13-62)
[channel.pattern.getimage\(\)](#) (on page 13-62)
[channel.pattern.setimage\(\)](#) (on page 13-63)
[channel.pattern.snapshot\(\)](#) (on page 13-66)
[channel.reset\(\)](#) (on page 13-68)
[channel.setbackplane\(\)](#) (on page 13-70)
[channel.setdelay\(\)](#) (on page 13-72)
[channel.setforbidden\(\)](#) (on page 13-73)
[channel.setlabel\(\)](#) (on page 13-74)
[channel.setpole\(\)](#) (on page 13-79)

dataqueue functions and attributes

[dataqueue.add\(\)](#) (on page 3-10)
[dataqueue.CAPACITY](#) (on page 3-10)
[dataqueue.clear\(\)](#) (on page 3-10)

[dataqueue.count](#) (on page 3-10)
[dataqueue.next\(\)](#) (on page 3-11)

delay functions

[delay\(\)](#) (on page 13-86)

digio functions and attributes

[digio.readbit\(\)](#) (on page 13-87)
[digio.readport\(\)](#) (on page 13-87)
[digio.trigger\[N\].assert\(\)](#) (on page 13-88)
[digio.trigger\[N\].clear\(\)](#) (on page 13-88)
[digio.trigger\[N\].mode](#) (on page 13-88)
[digio.trigger\[N\].overrun](#) (on page 13-89)
[digio.trigger\[N\].pulsewidth](#) (on page 13-90)
[digio.trigger\[N\].release\(\)](#) (on page 13-90)
[digio.trigger\[N\].stimulus](#) (on page 13-91)
[digio.trigger\[N\].wait\(\)](#) (on page 13-92)
[digio.writebit\(\)](#) (on page 13-92)
[digio.writeport\(\)](#) (on page 13-92)
[digio.writeprotect](#) (on page 13-93)

display functions and attributes

[display.clear\(\)](#) (on page 13-93)
[display.getannunciators\(\)](#) (on page 13-94)
[display.getcursor\(\)](#) (on page 13-95)
[display.getlastkey\(\)](#) (on page 13-96)
[display.gettext\(\)](#) (on page 13-97)
[display.inputvalue\(\)](#) (on page 13-98)
[display.loadmenu.add](#) (on page 13-100)
[display.loadmenu.delete\(\)](#) (on page 13-101)
[display.locallockout](#) (on page 13-102)
[display.menu\(\)](#) (on page 13-102)
[display.prompt\(\)](#) (on page 13-102)
[display.screen](#) (on page 13-104)
[display.sendkey\(\)](#) (on page 13-105)
[display.setcursor\(\)](#) (on page 13-105)
[display.settext\(\)](#) (on page 13-106)
[display.waitkey\(\)](#) (on page 13-107)

dmm functions and attributes

[dmm.adjustment.count](#) (on page 13-109)
[dmm.adjustment.date](#) (on page 13-109)
[dmm.aperture](#) (on page 13-110)
[dmm.appendbuffer\(\)](#) (on page 13-111)
[dmm.autodelay](#) (on page 13-112)
[dmm.autorange](#) (on page 13-113)
[dmm.autozero](#) (on page 13-114)
[dmm.buffer.catalog\(\)](#) (on page 13-115)
[dmm.buffer.info\(\)](#) (on page 13-116)
[dmm.buffer.maxcapacity](#) (on page 13-117)
[dmm.buffer.usedcapacity](#) (on page 13-118)
[dmm.calibration.ac\(\)](#) (on page 13-118)
[dmm.calibration.dc\(\)](#) (on page 13-120)
[dmm.calibration.lock\(\)](#) (on page 13-121)
[dmm.calibration.password](#) (on page 13-122)

[dmm.calibration.save\(\)](#) (on page 13-122)
[dmm.calibration.unlock\(\)](#) (on page 13-122)
[dmm.calibration.verifydate](#) (on page 13-123)
[dmm.close\(\)](#) (on page 13-123)
[dmm.configure.catalog\(\)](#) (on page 13-125)
[dmm.configure.delete\(\)](#) (on page 13-125)
[dmm.configure.query\(\)](#) (on page 13-126)
[dmm.configure.recall\(\)](#) (on page 13-128)
[dmm.configure.set\(\)](#) (on page 13-129)
[dmm.connect](#) (on page 13-130)
[dmm.dbreference](#) (on page 13-131)
[dmm.detectorbandwidth](#) (on page 13-132)
[dmm.displaydigits](#) (on page 13-132)
[dmm.drycircuit](#) (on page 13-133)
[dmm.filter.count](#) (on page 13-134)
[dmm.filter.enable](#) (on page 13-134)
[dmm.filter.type](#) (on page 13-135)
[dmm.filter.window](#) (on page 13-136)
[dmm.fourrtd](#) (on page 13-137)
[dmm.func](#) (on page 13-137)
[dmm.getconfig\(\)](#) (on page 13-139)
[dmm.inputdivider](#) (on page 13-140)
[dmm.limit\[Y\].autoclear](#) (on page 13-141)
[dmm.limit\[Y\].clear\(\)](#) (on page 13-141)
[dmm.limit\[Y\].enable](#) (on page 13-142)
[dmm.limit\[Y\].high.fail](#) (on page 13-142)
[dmm.limit\[Y\].high.value](#) (on page 13-143)
[dmm.limit\[Y\].low.fail](#) (on page 13-143)
[dmm.limit\[Y\].low.value](#) (on page 13-144)
[dmm.linesync](#) (on page 13-144)
[dmm.makebuffer\(\)](#) (on page 7-8)
[dmm.math.enable](#) (on page 13-147)
[dmm.math.format](#) (on page 13-147)
[dmm.math.mxb.bfactor](#) (on page 13-148)
[dmm.math.mxb.mfactor](#) (on page 13-149)
[dmm.math.mxb.units](#) (on page 13-149)
[dmm.math.percent](#) (on page 13-150)
[dmm.measure\(\)](#) (on page 13-150)
[dmm.measurecount](#) (on page 13-151)
[dmm.measurewithtime\(\)](#) (on page 13-151)
[dmm.nplc](#) (on page 13-152)
[dmm.offsetcompensation](#) (on page 13-153)
[dmm.open\(\)](#) (on page 13-154)
[dmm.opendetector](#) (on page 13-155)
[dmm.range](#) (on page 13-156)
[dmm.refjunction](#) (on page 13-157)
[dmm.rel.acquire\(\)](#) (on page 13-158)
[dmm.rel.enable](#) (on page 13-159)
[dmm.rel.level](#) (on page 13-160)
[dmm.reset\(\)](#) (on page 13-161)
[dmm.rtdalpha](#) (on page 13-162)
[dmm.rtdbeta](#) (on page 13-163)
[dmm.rtddelta](#) (on page 13-164)
[dmm.rtdzero](#) (on page 13-165)
[dmm.savebuffer\(\)](#) (on page 7-10)
[dmm.setconfig\(\)](#) (on page 13-168)
[dmm.simreftemperature](#) (on page 13-169)
[dmm.thermistor](#) (on page 13-170)
[dmm.thermocouple](#) (on page 13-171)
[dmm.threertd](#) (on page 13-172)
[dmm.threshold](#) (on page 13-173)
[dmm.transducer](#) (on page 13-173)
[dmm.units](#) (on page 13-174)

errorqueue functions and attributes

[errorqueue.clear\(\)](#) (on page 13-176)
[errorqueue.count](#) (on page 13-176)
[errorqueue.next\(\)](#) (on page 13-176)

eventlog functions and attributes

[eventlog.all\(\)](#) (on page 13-178)
[eventlog.clear\(\)](#) (on page 13-178)
[eventlog.count](#) (on page 13-179)
[eventlog.enable](#) (on page 13-179)
[eventlog.next\(\)](#) (on page 13-180)

exit functions

[exit\(\)](#) (on page 13-180)

file functions

[file.close\(\)](#) (on page 9-10)
[file.flush\(\)](#) (on page 9-10)
[file.read\(\)](#) (on page 9-10)
[file.seek\(\)](#) (on page 9-11)
[file.write\(\)](#) (on page 9-11)

format attributes

[format.asciiprecision](#) (on page 13-183)
[format.byteorder](#) (on page 13-183)
[format.data](#) (on page 13-184)

fs functions

[fs.chdir\(\)](#) (on page 9-9)
[fs.cwd\(\)](#) (on page 9-9)
[fs.is_dir\(\)](#) (on page 9-9)
[fs.is_file\(\)](#) (on page 9-9)
[fs.mk_dir\(\)](#) (on page 9-9)
[fs.readdir\(\)](#) (on page 9-9)
[fs.rmdir\(\)](#) (on page 9-10)

gpib attributes

[gpib.address](#) (on page 13-187)

io functions

[io.close\(\)](#) (on page 9-12)
[io.flush\(\)](#) (on page 9-12)
[io.input\(\)](#) (on page 9-12)
[io.open\(\)](#) (on page 9-13)
[io.output\(\)](#) (on page 9-13)
[io.read\(\)](#) (on page 9-13)
[io.type\(\)](#) (on page 9-14)
[io.write\(\)](#) (on page 9-14)

LAN functions and attributes

[lan.applysettings\(\)](#) (on page 13-190)
[lan.config.autonegotiate](#) (on page 13-191)
[lan.config.dns.address\[index\]](#) (on page 13-191)
[lan.config.dns.domain](#) (on page 13-192)
[lan.config.dns.dynamic](#) (on page 13-193)
[lan.config.dns.hostname](#) (on page 13-193)
[lan.config.dns.verify](#) (on page 13-194)
[lan.config.duplex](#) (on page 13-194)
[lan.config.gateway](#) (on page 13-195)
[lan.config.ipaddress](#) (on page 13-195)
[lan.config.method](#) (on page 13-196)
[lan.config.speed](#) (on page 13-196)

[lan.config.subnetmask](#) (on page 13-197)
[lan.lxidomain](#) (on page 13-197)
[lan.pingenable](#) (on page 13-198)
[lan.status.dns.address\[N\]](#) (on page 13-199)
[lan.status.dns.hostname](#) (on page 13-200)
[lan.status.duplex](#) (on page 13-200)
[lan.status.gateway](#) (on page 13-200)
[lan.status.ipaddress](#) (on page 13-201)
[lan.status.macaddress](#) (on page 13-201)
[lan.status.port.dst](#) (on page 13-201)
[lan.status.port.rawsocket](#) (on page 13-202)
[lan.status.port.telnet](#) (on page 13-202)
[lan.status.port.vxill](#) (on page 13-202)
[lan.status.reset\(\)](#) (on page 13-202)
[lan.status.speed](#) (on page 13-203)
[lan.status.subnetmask](#) (on page 13-203)
[lan.trigger\[N\].assert\(\)](#) (on page 13-203)
[lan.trigger\[N\].clear\(\)](#) (on page 13-204)
[lan.trigger\[N\].connect\(\)](#) (on page 13-204)
[lan.trigger\[N\].connected](#) (on page 13-204)
[lan.trigger\[N\].EVENT_ID](#) (on page 13-204)
[lan.trigger\[N\].ipaddress](#) (on page 13-204)
[lan.trigger\[N\].mode](#) (on page 13-205)
[lan.trigger\[N\].overrun](#) (on page 13-206)
[lan.trigger\[N\].protocol](#) (on page 13-207)
[lan.trigger\[N\].pseudostate](#) (on page 13-207)
[lan.trigger\[N\].stimulus](#) (on page 13-208)
[lan.trigger\[N\].wait\(\)](#) (on page 13-209)

localnode functions and attributes

[localnode.define.*](#) (on page 13-210)
[localnode.description](#) (on page 13-210)
[localnode.execute\(\)](#) (on page 3-11)
[localnode.getglobal\(\)](#) (on page 3-11)
[localnode.linefreq](#) (on page 13-211)
[localnode.model](#) (on page 13-211)
[localnode.password](#) (on page 13-212)
[localnode.passwordmode](#) (on page 13-212)
[localnode.prompts](#) (on page 13-212)
[localnode.reset\(\)](#) (on page 13-213)
[localnode.revision](#) (on page 13-214)
[localnode.serialno](#) (on page 13-214)
[localnode.setglobal\(\)](#) (on page 3-12)
[localnode.settime\(\)](#) (on page 13-215)
[localnode.setup.poweron](#) (on page 13-215)
[localnode.setup.recall\(\)](#) (on page 13-216)
[localnode.setup.save\(\)](#) (on page 13-216)
[localnode.showerrors](#) (on page 13-217)

makegetter functions

[makegetter\(\)](#) (on page 13-218)
[makesetter\(\)](#) (on page 13-218)

memory functions

[memory.available\(\)](#) (on page 13-219)
[memory.used\(\)](#) (on page 13-219)

opc functions

[opc\(\)](#) (on page 13-220)

printbuffer

[printbuffer\(\)](#) (on page 13-221)
[printnumber\(\)](#) (on page 13-222)

ptp functions and attributes

[ptp.burst.enable\(\)](#) (on page 13-223)
[ptp.ds.current\(\)](#) (on page 13-223)
[ptp.ds.default\(\)](#) (on page 13-224)
[ptp.ds.foreignmaster\(\)](#) (on page 13-224)
[ptp.ds.globaltime\(\)](#) (on page 13-225)
[ptp.ds.parent\(\)](#) (on page 13-225)
[ptp.ds.portconfig\(\)](#) (on page 13-226)
[ptp.enable\(\)](#) (on page 13-227)
[ptp.portstate](#) (on page 13-227)
[ptp.preferredmaster.enable\(\)](#) (on page 13-228)
[ptp.subdomain](#) (on page 13-228)
[ptp.synchronized](#) (on page 13-228)
[ptp.syncinterval](#) (on page 13-229)
[ptp.time](#) (on page 13-229)
[ptp.utcoffset](#) (on page 13-229)

reset functions

[reset\(\)](#) (on page 13-230)

scan functions and attributes

[scan.abort\(\)](#) (on page 13-230)
[scan.add\(\)](#) (on page 13-230)
[scan.background\(\)](#) (on page 13-232)
[scan.bypass](#) (on page 13-233)
[scan.create\(\)](#) (on page 13-234)
[scan.execute\(\)](#) (on page 13-236)
[scan.list\(\)](#) (on page 13-237)
[scan.measurecount](#) (on page 13-238)
[scan.mode](#) (on page 13-239)
[scan.nobufferbackground\(\)](#) (on page 13-240)
[scan.nobufferexecute\(\)](#) (on page 13-241)
[scan.reset\(\)](#) (on page 13-242)
[scan.scancount](#) (on page 13-242)
[scan.state\(\)](#) (on page 13-243)
[scan.stepcount](#) (on page 13-243)
[scan.trigger.arm.clear\(\)](#) (on page 13-244)
[scan.trigger.arm.set\(\)](#) (on page 13-244)
[scan.trigger.arm.stimulus](#) (on page 13-244)
[scan.trigger.channel.clear\(\)](#) (on page 13-245)
[scan.trigger.channel.set\(\)](#) (on page 13-245)
[scan.trigger.channel.stimulus](#) (on page 13-246)
[scan.trigger.clear\(\)](#) (on page 13-247)
[scan.trigger.measure.clear\(\)](#) (on page 13-247)
[scan.trigger.measure.set\(\)](#) (on page 13-247)
[scan.trigger.measure.stimulus](#) (on page 13-247)
[scan.trigger.sequence.clear\(\)](#) (on page 13-248)
[scan.trigger.sequence.set\(\)](#) (on page 13-248)

13-248)
[scan.trigger.sequence.stimulus](#) (on page 13-249)

schedule functions and attributes

[schedule.alarm\[x\].enable](#) (on page 13-250)
[schedule.alarm\[x\].EVENT_ID](#) (on page 13-250)
[schedule.alarm\[x\].fractionalseconds](#) (on page 13-250)
[schedule.alarm\[x\].period](#) (on page 13-251)
[schedule.alarm\[x\].ptpseconds](#) (on page 13-251)
[schedule.alarm\[x\].repetition](#) (on page 13-251)
[schedule.alarm\[x\].seconds](#) (on page 13-251)
[schedule.disable\(\)](#) (on page 13-252)

setup functions and attributes

[setup.cards\(\)](#) (on page 13-252)
[setup.poweron](#) (on page 13-253)
[setup.recall\(\)](#) (on page 13-253)
[setup.save\(\)](#) (on page 13-254)

slot[X] attributes

[slot\[X\].commonsideohms](#) (on page 13-255)
[slot\[X\].digio](#) (on page 13-255)
[slot\[X\].endchannel.amps](#) (on page 13-255)
[slot\[X\].endchannel.isolated](#) (on page 13-256)
[slot\[X\].endchannel.voltage](#) (on page 13-257)
[slot\[X\].idn](#) (on page 13-257)
[slot\[X\].interlock.override](#) (on page 13-257)
[slot\[X\].interlock.state](#) (on page 13-258)
[slot\[X\].isolated](#) (on page 13-259)
[slot\[X\].matrix](#) (on page 13-259)
[slot\[X\].maxvoltage](#) (on page 13-259)
[slot\[X\].multiplexer](#) (on page 13-259)
[slot\[X\].poles.four](#) (on page 13-260)
[slot\[X\].poles.one](#) (on page 13-260)
[slot\[X\].poles.two](#) (on page 13-260)
[slot\[X\].pseudocard](#) (on page 13-260)
[slot\[X\].startchannel.amps](#) (on page 13-261)
[slot\[X\].startchannel.isolated](#) (on page 13-262)
[slot\[X\].startchannel.voltage](#) (on page 13-263)
[slot\[X\].tempsensor](#) (on page 13-263)
[slot\[X\].thermal.state](#) (on page 13-263)

status functions and attributes

[status.condition](#) (on page 13-264)
[status.measurement.*](#) (on page 13-265)
[status.node_enable](#) (on page 13-267)
[status.node_event](#) (on page 13-269)
[status.operation.*](#) (on page 13-270)
[status.operation.user.*](#) (on page 13-271)
[status.questionable.*](#) (on page 13-273)
[status.request_enable](#) (on page 13-276)

[status.request_event](#) (on page 13-278)
[status.reset\(\)](#) (on page 13-278)
[status.standard.*](#) (on page 13-278)
[status.system.*](#) (on page 13-280)
[status.system2.*](#) (on page 13-281)
[status.system3.*](#) (on page 13-282)
[status.system4.*](#) (on page 13-284)
[status.system5.*](#) (on page 13-285)

timer functions

[timer.measure.t\(\)](#) (on page 13-286)
[timer.reset\(\)](#) (on page 13-287)

trigger functions and attributes

[trigger.blender\[N\].clear\(\)](#) (on page 13-287)
[trigger.blender\[N\].orenable](#) (on page 13-287)
[trigger.blender\[N\].overrun](#) (on page 13-288)
[trigger.blender\[N\].stimulus\[M\]](#) (on page 13-288)
[trigger.blender\[N\].wait\(\)](#) (on page 13-289)
[trigger.clear\(\)](#) (on page 13-289)
[trigger.wait\(\)](#) (on page 13-290)

trigger.timer functions and attributes

[trigger.timer\[N\].clear](#) (on page 13-290)
[trigger.timer\[N\].count](#) (on page 13-290)
[trigger.timer\[N\].delay](#) (on page 13-291)
[trigger.timer\[N\].delaylist](#) (on page 13-291)
[trigger.timer\[N\].overrun](#) (on page 13-292)
[trigger.timer\[N\].passthrough](#) (on page 13-292)
[trigger.timer\[N\].triggerstimulus](#) (on page 13-292)
[trigger.timer\[N\].wait\(\)](#) (on page 13-293)

tsplink functions and attributes

[tsplink.group](#) (on page 3-12)
[tsplink.master](#) (on page 3-12)
[tsplink.node](#) (on page 13-294)
[tsplink.reset\(\)](#) (on page 13-294)
[tsplink.state](#) (on page 13-295)

tsplink.trigger functions and attributes

[tsplink.trigger\[N\].assert\(\)](#) (on page 3-12)
[tsplink.trigger\[N\].clear\(\)](#) (on page 3-13)
[tsplink.trigger\[N\].mode](#) (on page 3-13)
[tsplink.trigger\[N\].overrun](#) (on page 3-15)
[tsplink.trigger\[N\].release\(\)](#) (on page 3-15)
[tsplink.trigger\[N\].stimulus](#) (on page 13-298)
[tsplink.trigger\[N\].wait\(\)](#) (on page 3-15)

tspnet functions and attributes

[tspnet.clear\(\)](#) (on page 10-10)
[tspnet.connect\(\)](#) (on page 10-5)
[tspnet.disconnect\(\)](#) (on page 10-10)
[tspnet.execute\(\)](#) (on page 10-6)
[tspnet.idn\(\)](#) (on page 10-6)
[tspnet.read\(\)](#) (on page 10-8)
[tspnet.readavailable\(\)](#) (on page 10-9)
[tspnet.reset\(\)](#) (on page 10-10)
[tspnet.termination\(\)](#) (on page 10-11)
[tspnet.timeout](#) (on page 10-11)
[tspnet.tsp.abort\(\)](#) (on page 10-13)
[tspnet.tsp.abortonconnect](#) (on page 10-14)
[tspnet.tsp.rhtablecopy\(\)](#) (on page 10-12)
[tspnet.tsp.runscript\(\)](#) (on page 10-12)
[tspnet.write\(\)](#) (on page 10-7)

upgrade functions

[upgrade.previous\(\)](#) (on page 13-309)
[upgrade.unit\(\)](#) (on page 13-309)

userstring functions

[userstring.add\(\)](#) (on page 13-310)
[userstring.catalog\(\)](#) (on page 13-310)
[userstring.delete\(\)](#) (on page 13-311)
[userstring.get\(\)](#) (on page 13-311)

waitcomplete functions

[waitcomplete\(\)](#) (on page 3-16)

beeper functions and attributes

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	<code>beeper.beep(duration, frequency)</code> 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 (on page 13-16)). This function is an overlapped command. Script execution will continue and not wait for the beep to finish. If another beep command is sent before the previous beep finishes, the first beep is terminated. You can use the function waitcomplete() (on page 3-16) to pause script execution until the beep command finishes.
Also see	beeper.enable (on page 13-16)
Example	Enables the beeper and generates a two-second, 2400Hz beep: <pre>beeper.enable = 1 beeper.beep(2, 2400)</pre>

beeper.enable	
Attribute	Beeper control (on/off).
Usage	To read the state of the beeper: <code>beeperstate = beeper.enable</code> To write the state of the beeper: <code>beeper.enable = beeperstate</code> Set beeperstate to one of the following values: <ul style="list-style-type: none"> • 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() (on page 13-16)
Example	Enables the beeper and generates a two-second, 2400Hz beep: <pre>beeper.enable = 1 beeper.beep(2, 2400)</pre>

bit functions

Logic and bit operations

The bit functions are used to perform bitwise logic operations on two given numbers, and bit operations on one given number. Logic and bit operations truncate the fractional part of given numbers to make them integers.

NOTE The Test Script Processor (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	<pre>value = bit.bitand(value1, value2)</pre> <p>value1: First number for the AND operation. value2: Second number for the AND operation. value: Returned result of the AND operation.</p>
Remarks	<ul style="list-style-type: none"> This function performs a logical AND operation on two numbers. Any fractional parts of <code>value1</code> and <code>value2</code> are truncated to make them integers. The returned value is also an integer.

bit.bitand()	
Also see	Logic and bit operations (on page 13-17) bit.bitor() (on page 13-18) bit.bitxor() (on page 13-18)
Example	AND'ing decimal 10 (binary 1010) with decimal 9 (binary 1001) will return a value of decimal 8 (binary 1000): <pre>value = bit.bitand(10, 9) print(value)</pre> Output: 8.000000e+00

bit.bitor()	
Function	Performs a bitwise logical OR operation on two numbers.
Usage	<pre>value = bit.bitor(value1, value2)</pre> <p>value1: First number for the OR operation. value2: Second number for the OR operation. value: Returned result of the OR operation.</p>
Remarks	<ul style="list-style-type: none"> This function performs a logical OR operation on two numbers. Any fractional parts of <code>value1</code> and <code>value2</code> are truncated to make them integers. The returned value is also an integer.
Also see	Logic and bit operations (on page 13-17) bit.bitand() (on page 13-17) bit.bitxor() (on page 13-18)
Example	Performs a bitwise logical OR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001); will return a value of decimal 11 (binary 1011): <pre>value = bit.bitor(10, 9) print(value)</pre> Output: 1.100000e+01

bit.bitxor()	
Function	Performs a bitwise logical XOR (Exclusive OR) operation on two numbers.
Usage	<pre>value = bit.xor(value1, value2)</pre> <p>value1: First number for the XOR operation. value2: Second number for the XOR operation. value: Returned result of the XOR operation.</p>
Remarks	<ul style="list-style-type: none"> This function performs a logical Exclusive OR operation on two numbers. Any fractional parts of <code>value1</code> and <code>value2</code> are truncated to make them integers. The returned value is also an integer.

bit.bitxor()	
Details	Logic and bit operations (on page 13-17) bit.bitxor() (on page 13-18)
Example	Performs a bitwise logical exclusive OR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001); will return a value of decimal 3 (binary 0011): <pre>value = bit.bitxor(10, 9) print(value)</pre> Output: 3.000000e+00

bit.clear()	
Function	Clears a bit at a given index position.
Usage	<pre>value = bit.clear(value1, index)</pre> value1: Given number. index: Index position of the bit to be cleared (1 to 32). value: Returns the result of the manipulation.
Remarks	<ul style="list-style-type: none"> This function clears a bit at a given index position. Any fractional part of <code>value1</code> 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.
Also see	Logic and bit operations (on page 13-17) bit.get() (on page 13-19) bit.getfield() (on page 13-20) bit.set() (on page 13-21) bit.setfield() (on page 13-21) bit.toggle() (on page 13-23)
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): <pre>value = bit.clear(15, 2) print(value)</pre> Output: 1.300000e+01

bit.get()	
Function	Retrieves the weighted value of a bit at a given index position.
Usage	<pre>value = bit.get(value1, index)</pre> value1: Given number. index: Index position of the bit to be retrieved (1 to 32). value: Returned weighted value of the bit.

bit.get()	
Remarks	<ul style="list-style-type: none"> • This function returns the value of the bit in <code>value1</code> at the given index. This is the same as returning <code>value1</code> with all other non-indexed bits set to zero. • 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 13-17) for more information.
Also see	bit.clear() (on page 13-19) bit.set() (on page 13-21) bit.setfield() (on page 13-21) bit.test() (on page 13-22) bit.toggle() (on page 13-23)
Example	<p>The binary equivalent of decimal 10 is 1010. Getting the bit at index position 4 will return decimal value 8:</p> <pre>value = bit.get(10, 4) print(value)</pre> <p>Output: 8.000000e+00</p>

bit.getfield()	
Function	Returns a field of bits starting at a given index position.
Usage	<pre>value = bit.getfield(value1, index, width)</pre> <p>value1: Given number. index: Index position of the first bit; 1 to (33 - width). width: Field width - number of bits to be included in the field; 1 to 24. value: Returned value of the bit field.</p>
Remarks	<ul style="list-style-type: none"> • A field of bits is a contiguous group of bits. This function retrieves a field of bits from <code>value1</code>, 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 <code>width</code>. • 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.

bit.getfield()	
Also see	<p>Logic and bit operations (on page 13-17)</p> <p>bit.clear() (on page 13-19)</p> <p>bit.get() (on page 13-19)</p> <p>bit.set() (on page 13-21)</p> <p>bit.setfield() (on page 13-21)</p> <p>bit.test() (on page 13-22)</p> <p>bit.toggle() (on page 13-23)</p>
Example	<p>The binary equivalent of decimal 13 is 1101. The field at index 2 and width 3 consists of the binary bits 110. The returned value will be decimal 6 (binary 110):</p> <pre>value = bit.getfield(13, 2, 3)</pre> <p>Output: 6.000000e+00</p>
bit.set()	
Function	Sets a bit at a given index position.
Usage	<pre>value = bit.set(value1, index)</pre> <p>value1: Given number.</p> <p>index: Index position of the bit to be set (1 to 32).</p> <p>value: Returned value of the new number.</p>
Remarks	<ul style="list-style-type: none"> This function returns <code>value</code>, which is <code>value1</code> 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 <code>value1</code> will be truncated to make it an integer.
Also see	<p>Logic and bit operations (on page 13-17)</p> <p>bit.clear() (on page 13-19)</p> <p>bit.get() (on page 13-19)</p> <p>bit.getfield() (on page 13-20)</p> <p>bit.setfield() (on page 13-21)</p> <p>bit.test() (on page 13-22)</p> <p>bit.toggle() (on page 13-23)</p>
Example	<p>The binary equivalent of decimal 8 is 1000. If the bit at index 3 is set to 1, the returned value will be decimal 12 (binary 1100):</p> <pre>value = bit.set(8, 3)</pre> <p>Output: 1.200000e+01</p>

bit.setfield()	
Function	Overwrites a bit field at a given index position.
Usage	<pre>value = bit.setfield(value1, index, width, fieldvalue)</pre> <p>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.</p>
Remarks	<ul style="list-style-type: none"> • This function returns <code>value</code>, which is <code>value1</code> 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 <code>fieldvalue</code>. The least significant bit in <code>value1</code> has an index of 1 and the most significant bit has an index of 32. • Prior to setting the field of bits, any fractional parts of <code>value1</code> and <code>fieldvalue</code> will be truncated to make them integers. • If the <code>fieldvalue</code> is wider than the <code>width</code>, the extra most significant bits of the <code>fieldvalue</code> will be truncated. For example, assume the width is 4 bits, and the binary value for <code>fieldwidth</code> is 11110 (5 bits). The most significant bit of <code>fieldwidth</code> will be truncated, and a binary value of 1110 will be used as the <code>fieldvalue</code>.
Also see	<p>Logic and bit operations (on page 13-17)</p> <p>bit.clear() (on page 13-19)</p> <p>bit.getfield() (on page 13-20)</p> <p>bit.set() (on page 13-21)</p> <p>bit.test() (on page 13-22)</p> <p>bit.toggle() (on page 13-23)</p>
Example	<p>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):</p> <pre>value = bit.setfield(15, 2, 3, 5) print(value)</pre> <p>Output: 1.100000e+01</p>

bit.test()	
Function	Returns the Boolean value (true or false) of a bit at a given index position.
Usage	<pre>value = bit.test(value1, index)</pre> <p>value1: Given number. index: Index position of the bit to be tested (1 to 32). value: Returned decimal value of the bit.</p>

bit.test()	
Remarks	<ul style="list-style-type: none"> • This function returns <code>value</code>, which is the result of the tested bit. The least significant bit of the given number is at index 1. The most significant bit is at index 32. • Any fractional part of <code>value1</code> will be truncated to make it an integer. If the indexed bit for <code>value1</code> is set to 0, the returned value will be false. If the indexed bit for <code>value1</code> is set to 1, the returned value will be true. • If the index is bigger than the number of bits in <code>value1</code>, the result will be false.
Also see	<p>Logic and bit operations (on page 13-17)</p> <p>bit.clear() (on page 13-19)</p> <p>bit.get() (on page 13-19)</p> <p>bit.getfield() (on page 13-20)</p> <p>bit.set() (on page 13-21)</p> <p>bit.setfield() (on page 13-21)</p> <p>bit.toggle() (on page 13-23)</p>
Example	<p>The binary equivalent of decimal 10 is 1010. Testing the bit at index position 4 will return a Boolean value of true:</p> <pre>value = bit.test(10, 4) print(value)</pre> <p>Output: true</p>

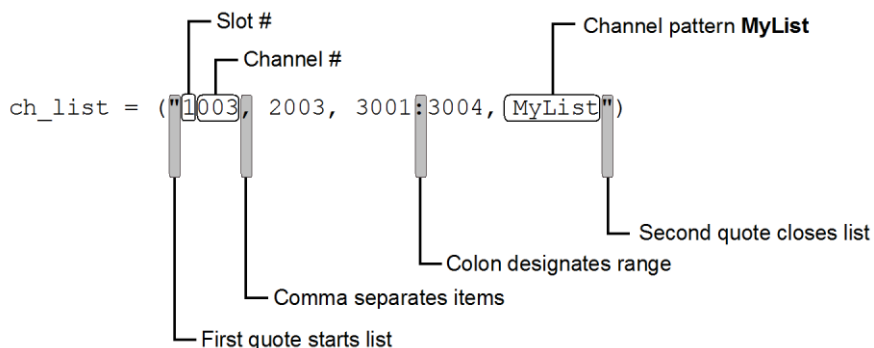
bit.toggle()	
Function	Toggles the value of a bit at a given index position.
Usage	<pre>value = bit.toggle(value1, index)</pre> <p>value1: Given number.</p> <p>index: Index position of the bit to be toggled (1 to 32).</p> <p>value: Returned value of the new number.</p>
Remarks	<ul style="list-style-type: none"> • This function returns <code>value</code>, which is the result of toggling a bit in <code>value1</code>. • Any fractional part of <code>value1</code> is truncated to make it an integer. The returned decimal value is also an integer. The least significant bit of the given number is index 1. The most significant bit is index 32. • The indexed bit for <code>value1</code> is toggled from 0 to 1, or 1 to 0.

bit.toggle()	
Also see	Logic and bit operations (on page 13-17) bit.clear() (on page 13-19) bit.get() (on page 13-19) bit.getfield() (on page 13-20) bit.set() (on page 13-21) bit.setfield() (on page 13-21) bit.test() (on page 13-22)
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). <pre>value = bit.toggle(10, 3) print(value)</pre> Output: 1.400000e+01

channel functions and attributes

Use the functions and attributes in this group to control and query switching channels. Unless specifically noted, <ch_list> specifies the channels, backplane relays, and channel patterns in a comma-delimited format on which the function is to be performed. The following figure shows this format:

Figure 13-1: ch_list legend



There are three different notations used to control relays: Backplane relay notation, MUX (multiplexer) channel notation, and Matrix card notation.

To control analog backplane relays for slots with analog backplane relay channels, use $S9BX$, where:

S: Slot number

9: Backplane notation designation (always 9 when referencing a backplane relay)

B: Bank number

X: Analog backplane relay number

Analog backplane relays (Bank 2 of Slot 1) examples:

Reference	Analog backplane relay
1921	analog backplane relay 1
1922	analog backplane relay 2
1923	analog backplane relay 3
1924	analog backplane relay 4
1925	analog backplane relay 5
1926	analog backplane relay 6

To control channels using MUX channel notation, use $SCCC$, where:

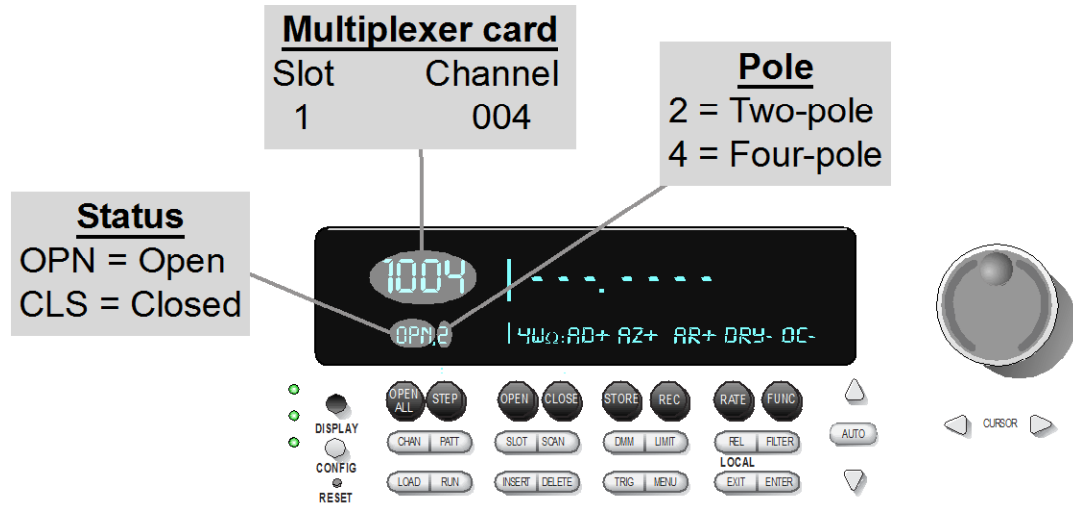
S: Slot number

CCC: Channel number (always use 3 digits)

Multiplexer examples:

Reference	Slot	Channel
1004	1	004
1020	1	020
2100	2	100
3003	3	003

Figure 13-2: Multiplexer card display



To control channels using matrix card notation, use SRCC, where:

S: Slot number

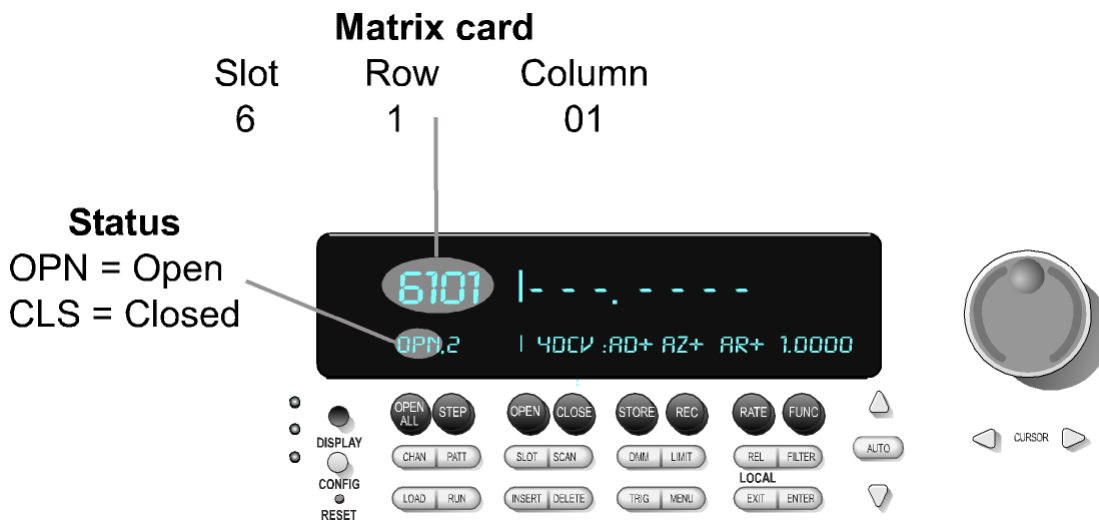
R: Row number

CC: Column number (always use 2 digits)

Matrix channel examples:

Reference	Slot	Row	Column
1104	1	1	04
1203	1	2	03
2305	2	3	05
3112	3	1	12
6101	6	1	01

Figure 13-3: Matrix card display



Using channel.*() ICL commands

Unless otherwise noted, `channel.*()` Instrument Control Library (ICL) commands use the channel list and return the value syntax described below.

- The channel list is specified according to the syntax presented in the channel list legend. Not all ICL commands support the fully described syntax. Any exclusions are noted in a specific command's documentation.
- There are five different types of channels available on the supported Model 3706 cards. These include switch (or relay), backplane, totalizer, DAC, and digital I/O. Even though the channels are specified in an identical manner, not all ICL commands act on all channel types. The descriptions of each ICL command provide more information.
- When acting on a range of channels is necessary or more convenient, use the ':' notation. For example, to specify Channels 1 through 20 on Slot 4, use `4001:4020`.

```
print(channel.getlabel("4001:4020"))
```
- When acting on an entire slot is necessary or more convenient, use the `slotX` notation. For example, to specify all channels on Slot 4, use `slot4`.

```
print(channel.getlabel("slot4"))
```
- When acting on an entire instrument is necessary or more convenient, use the `allslots` notation. For example, to specify channels 1 through 20 on Slot 4, use `allslots`.

```
print(channel.getlabel("allslots"))
```
- When a range (including `slotX` and `allslots` notation) includes mixed channel types, the invalid channel types are ignored. If an invalid channel type is individually specified, then an error is generated.

The following errors can occur because of invalid channel list syntax or specification.

Error Message	Description
<code>invalid specified channel</code>	The channel is specified with the correct syntax, but does not exist on the card.
<code>invalid character in channel list</code>	The channel list contains an invalid character or syntax sequence.
<code>invalid slot in channel list</code>	The slot specified in the channel list is empty.
<code>invalid channel type in channel list</code>	The channel is specified with the correct syntax, but the channel type is not supported by the specified ICL command.
<code>no valid channels in channel list</code>	After processing, no valid channels remain in the command to act upon.

Error Message	Description
invalid label or pattern name	A string was found in the channel list that does not specify any known label or pattern name.
no patterns accepted	A pattern was specified for an ICL command that does not support patterns as input.
no multiple channels accepted	Multiple channels were specified for an ICL command that acts only on a single channel.
no range specifier accepted	A range was specified for an ICL command that does not support a range as input.
no slot specifier accepted	An entire slot was specified using slotX (for example, slot1) for an ICL command that does not support slotX as input.
no all slots specifier accepted	All slots were specified using allslots for an ICL command that does not support allslots as input.
no labels accepted	A label was specified for an ICL command that does not support labels as input.
no paired channels accepted	A channel was specified for an ICL command that does not act on paired channels.
no single channels accepted	A single channel was specified for an ICL command that only supports acting on groups of channels.
no multiple specifiers accepted	Multiple descriptions were specified for an ICL command that does not support multiple descriptions in a list.
channels all must be of same type	The provided channel list contains channels or various channel types, but the ICL command supports only channel lists that contain a single, consistent channel type.
forbidden channel	The channel specified is forbidden to be closed.

Return value

The return value is a string containing a list of comma-delimited individual return items. The channel list argument of the ICL command determines the number and order of these returned items.

When the channel list parameter for this function is 'slotX', the response first lists the channels starting from lowest to highest. After the channels, backplane relays are listed starting with the lowest bank first and increasing to the highest. More specifically, the channels are returned in numerical order.

When the channel list parameter for this function is 'allslots', the response starts with Slot 1 and increases to Slot 6. Each slot is processed completely before going to the next. Therefore, all Slot 1 channels and backplane relays are listed before Slot 2 channels.

When the response is numerical, but in string format, use the `tonumber()` function to convert a number string to a variable. For example,

```
TSP> x = tonumber("34.3")
TSP> print(x)
3.43e+001
```

When the response is a comma-delimited string, the individual numbers can be identified by iterating through the list using the comma delimiters. For example, the Lua code below will start at the beginning of a string and use the comma as a signal to break the string into small chunks. The `tonumber()` function is used on each chunk to isolate and convert the individual number.

```
s1 = 1
s2 = 1
e = string.len(text)
while s2 ~= e do
    s2 = string.find(text, ",", s1)
    if not s2 then s2 = e end
    print(tonumber(string.sub(text, s1, s2-1)))
    s1 = s2 + 1
end
```

channel.calibration.adjustcount()	
Function	Gets the number of times that a card has been calibrated.
Usage	<pre><mycount> = channel.calibration.adjustcount([ch_list])</pre> <p>mycount: Return value representing the number of times unit has been adjusted.</p> <p>ch_list: A string representing the slot holding the card to query.</p>
Remarks	<p>You can use this command with the instrument/channels either locked or unlocked. If no channel list is provided, the currently unlocked channels are assumed.</p> <p>There is only one adjustment count per card. Therefore, with no channel unlocked, the only acceptable values for channel list are <code>slot1</code>, <code>slot2</code>, and so on. Otherwise, an error is generated.</p>

channel.calibration.adjustcount()	
Example	To query for the adjustment count: <pre>CalCount = channel.calibration.adjustcount("1010")</pre>
channel.calibration.adjustdate()	
Function	Sets or gets the adjustment date in UTC format (number of seconds since January 1, 1970) on the unlocked channel.
Usage	<pre>mydate = channel.calibration.adjustdate([ch_list], [date])</pre> mydate: Return value representing the number of seconds since January 1, 1970. ch_list: A string representing the slot holding the card to query. date: Represents the number of seconds since January 1, 1970.
Remarks	This command can get the adjust date whether calibration is currently locked or unlocked. If the channel list is not specified, it uses the currently unlocked card. This command can only set the adjust date on a previously unlocked card. The date is not permanently saved until <code>channel.calibration.save()</code> (on page 13-32) is issued. There is only one adjustment date per card. Therefore, with no channel unlocked, the only acceptable values for channel list are <code>slot1</code> , <code>slot2</code> , and so on. Otherwise, an error is generated.
Also see	channel.calibration.save() (on page 13-32)
Example	To get the number of seconds since January 1, 1970: <pre>date = channel.calibration.adjustdate()</pre> To set the calibration adjustment date on the currently unlocked slot based on the current date of the system: <pre>channel.calibration.adjustdate(os.time())</pre> To set the calibration adjustment date on Slot 1 to July 4, 2006: <pre>channel.calibration.adjustdate(os.time(year=2006, month=7, day = 4))</pre> See Lua documentation for formatting options with <code>os.date()</code> and additional information on <code>os.time()</code> . <hr/> NOTE: The following example assumes the set date is July 4, 2006. <hr/> To query calibration adjustment date and format the response as mm/dd/yyyy: <pre>TSP> print(os.date("%m/%d/%Y", channel.calibration.adjustdate()))</pre> 07/04/2006
channel.calibration.lock()	
Function	Locks calibration on the card being calibrated.
Usage	<pre>channel.calibration.lock()</pre>

channel.calibration.lock()	
Remarks	<p>This command locks calibration on the card being calibrated, but does not save calibration data (which is lost if it is not saved before locking). Once locked, you must unlock calibration to perform it again.</p> <p>Only one card can be calibrated at a time. Therefore, the <code>lock</code> function works only on the currently unlocked card.</p> <p>An error is generated if this command is issued when calibration is already locked.</p>
Also see	channel.calibration.save() (on page 13-32)
Example	<p>To lock calibration:</p> <pre>channel.calibration.lock()</pre>

channel.calibration.password()	
Function	Sets the password needed to unlock the calibration of a card.
Usage	<pre>channel.calibration.password(<password>)</pre> <p>password: A string of characters that protects the command.</p>
Remarks	<p>There is only one password per card. Therefore, the <code>password</code> function works only on the currently unlocked card.</p> <p>Make note of the password, because there is no command to query for the password once it has been set on the system.</p> <p>This command generates an error if calibration is locked or if the password string length is greater than six characters. Passwords are alphanumeric and case-sensitive.</p> <p>The default password from the factory is KI3706. The first two characters in the password are capital K capital I (for Keithley Instruments).</p>
Also see	channel.calibration.unlock() (on page 13-34)
Example	<p>To set the password to "MyUnlock" after unlocking calibration with the saved password:</p> <pre>channel.calibration.password = "MyUnlock"</pre>

channel.calibration.save()	
Function	Saves calibration data after performing a calibration.
Usage	<pre>channel.calibration.save()</pre>

channel.calibration.save()	
Remarks	<p>Only one card can be calibrated at a time. Therefore, the <code>save</code> function works only on the currently unlocked card. An error is generated if this command is issued when calibration is already locked.</p> <p>The system must receive this command before the <code>channel.calibration.lock()</code> (on page 13-31) command or the data will be lost.</p> <p>This command saves the present values of the calibration constants and calibration date, and increases the calibration count by 1, regardless of errors in the data. You should not issue a <code>save</code> command unless the calibration procedure was performed with no errors. If no calibration date was specified (using <code>channel.calibration.*date</code>), then the date is auto-generated based on the system date.</p>
Also see	<p>channel.calibration.lock() (on page 13-31)</p> <p>channel.calibration.adjustdate() (on page 13-31)</p> <p>channel.calibration.verifydate() (on page 13-35)</p>
Example	<p>To save calibration data:</p> <pre>channel.calibration.save()</pre>

channel.calibration.step()	
Function	Sends a calibrate command to the card.
Usage	<pre>channel.calibration.step(<single channel>, <step>, [<value>])</pre> <p>single_channel: Channel to be calibrated.</p> <p>step: Number corresponding to the specified step.</p> <p>value: Measurement value for the particular step.</p>

channel.calibration.step()	
Remarks	<p>The specified channel must be on the unlocked slot. Only DAC and totalizer channels can be calibrated. It is best to calibrate a single channel sequentially to completion before changing channels.</p> <p>The card assumes that the given voltage or current value is exactly what it is sourcing for the given step. This command generates an error if the step is out of sequence, does not exist, or the calibration is locked. Also, an error is generated if the calibration step does not complete successfully, if the value passed is invalid or out of range for the step, or not needed.</p> <p>For DAC channels, a calibration sequence includes these steps:</p> <ol style="list-style-type: none"> 1. Set Voltage, -12 to +12 range, generate Negative Point 1. 2. Send reading. 3. Set Voltage, -12 to +12 range, generate Negative Point 2. 4. Send reading. 5. Set Voltage, -12 to +12 range, generate Positive Point 1. 6. Send reading. 7. Set Voltage, -12 to +12 range, generate Positive Point 2. 8. Send reading. 9. Set Current, 0 mA to +20 mA range, generate Point 1. 10. Send reading. 11. Set Current, 0 mA to +20 mA range, generate Point 2. 12. Send reading. 13. Set Current, +4 mA to +20 mA range, generate Point 1. 14. Send reading. 15. Set Current, +4 mA to +20 mA range, generate Point 2. 16. Send reading. <p>For totalizer channels, a calibration sequence includes these steps:</p> <ol style="list-style-type: none"> 1. Calibrate 0 V Totalizer Threshold 2. Calibrate 1.5 V Totalizer Threshold

channel.calibration.unlock()	
Function	Unlocks calibration.
Usage	<p><code>channel.calibration.unlock(<ch_list>, <password>)</code></p> <p>ch_list: A string representing the slot holding the card to query.</p> <p>password: A string of characters that protects the command.</p>

channel.calibration.unlock()	
Remarks	<p>There is only one password per card. Therefore, the only acceptable values for channel list are <code>slot1</code>, <code>slot2</code>, and so on. Otherwise, an error is generated.</p> <p>An error is generated if the password that is entered does not match the one that was saved with <code>channel.calibration.password()</code> (on page 13-32).</p> <p>The password can only contain six case-sensitive, alphanumeric characters.</p> <p>The default password from the factory is KI3706. The first two characters in the password are capital K capital I (for Keithley Instruments).</p>
Also see	channel.calibration.password() (on page 13-32)
Example	<p>To unlock calibration for first DAC using the default password:</p> <pre>channel.calibration.unlock("1010", "KI3706")</pre>
channel.calibration.verifydate()	
Function	Sets or gets the calibration verification date in UTC format (number of seconds since January 1, 1970).
Usage	<pre><mydate> = channel.calibration.verifydate([ch_list], [date])</pre> <p>mydate: Return value representing the number of seconds since January 1, 1970.</p> <p>ch_list: A string representing the slot holding the card to set or query.</p> <p>date: Represents the number of seconds since January 1, 1970.</p>
Remarks	<p>This command can get the verification date whether calibration is currently locked or unlocked. If the channel list is not specified, it uses the currently unlocked card.</p> <p>This command can only set the verification date on a previously unlocked card. The date is not permanently saved until <code>channel.calibration.save()</code> (on page 13-32) is issued.</p> <p>There is only one verification date per card. Therefore, with no channel unlocked, the only acceptable values for channel list are <code>slot1</code>, <code>slot2</code>, and so on. Otherwise, an error is generated.</p>
Also see	channel.calibration.save() (on page 13-32)

channel.calibration.verifydate()	
Example	<p>To query the number of seconds since January 1, 1970:</p> <pre>CalDate = channel.calibration.verifydate()</pre> <p>To set the calibration verification date on Slot 1 based on the current date of the system:</p> <pre>channel.calibration.verifydate(os.time())</pre> <p>To set the calibration verification date on Slot 1 as July 4, 2006:</p> <pre>channel.calibration.verifydate(os.time(year=2006, month=7, day = 4))</pre> <p>See Lua documentation for formatting options with <code>os.date()</code> and additional information on <code>os.time()</code>.</p> <hr/> <p>NOTE: Example assumes the set date is July 4, 2006.</p> <hr/> <p>To query calibration verification date and format the response as mm/dd/yyyy:</p> <pre>TSP> print(os.date("%m/%d/%Y", channel.calibration.verifydate("slot1")))</pre> <p>07/04/2006</p>
channel.clearforbidden()	
Function	Clears the list of channels specified from being forbidden to close.
Usage	<pre>channel.clearforbidden(<ch_list>)</pre> <p>ch_list: A string listing the items to no longer be forbidden to close.</p>
Remarks	<p>The <code>ch_list</code> parameter indicates the scope of channels affected and may include:</p> <ul style="list-style-type: none"> • <code>allslots</code> or <code>slotX</code> (where X equals 1 to 6). • Channel ranges or individual channels. • Analog backplane relays. <p>This function allows all items contained in the channel list parameter to be closed (removes the "forbidden to close" attribute that can be applied to a channel using channel.setforbidden() (on page 13-73)).</p> <p>An error will be generated if:</p> <ul style="list-style-type: none"> • The specified channel or analog backplane relay does not exist for card installed in a slot. • The specified channel or analog backplane relay is for an empty slot. • There is a parameter syntax error in the channel specified. <p>Command processing will stop as soon as an error is detected. If an error is found, the channels are not cleared from being forbidden to close. With no errors, the channels in the channel list parameter are cleared from being forbidden to close.</p>
Also see	<p>channel.getforbidden() (on page 13-50)</p> <p>channel.setforbidden() (on page 13-73)</p>

channel.clearforbidden()	
Example	To clear Channels 2, 4, 6, and 8 of Slot 2 from being forbidden to close: <pre>channel.clearforbidden('2002,2004,2006,2008')</pre> To clear all channels from being forbidden to close: <pre>channel.clearforbidden('allslots')</pre>
channel.close()	
Function	Closes specified items in channel list parameter without opening any channels.
Usage	<pre>channel.close(<ch_list>)</pre> ch_list: A string listing the channels and channel patterns to close.
Remarks	<p>This function closes channels and channel patterns (specified by <code>ch_list</code>). These closures are appended to the already closed channels (no previously closed channels are opened by this command).</p> <p>The <code>ch_list</code> parameter can include analog backplane relay items.</p> <p>For items specified in <code>ch_list</code>, this function closes the associated channels along with any associated analog backplane relays. For channel patterns, the analog backplane relays that get closed are the ones that were specified when the pattern was created (see <code>channel.pattern.setimage()</code> (on page 13-63) and <code>channel.pattern.snapshot()</code> (on page 13-66)). However, for channels, they are the ones specified with the <code>channel.setbackplane()</code> (on page 13-70) function. Another option for closing analog backplane relays with this command is to include them in the <code>ch_list</code> parameter.</p> <p>This command has no effect on how the DMM is configured.</p>

channel.close()	
	<p>Actions associated with this function include:</p> <ul style="list-style-type: none"> • Parse the parameter. • Close the specified items in <code>ch_list</code>. • Incur the settling time and user delay. • Command completes. <p>An error is generated if:</p> <ul style="list-style-type: none"> • Syntax error exists in parameter string. • An empty parameter string is specified, or a parameter string containing only spaces exists. • The parameter string contains 'slotX', where X = 1 to 6, or 'allslots'. • A specified channel or channel pattern is invalid. • Channel number does not exist for slot specified. • Slot is empty. • Channel pattern does not exist. • A forbidden item is specified. • Does not support being closed like a digital I/O channel. • Channel is paired with another bank for a multi-wire application. • Internal errors related to communication, power consumption, and so on. <p>Once an error is detected, the command stops processing and no channels are closed. Channels close only if no syntax errors exist in parameters and all channels are valid for closing.</p> <p>For digital I/O, DAC, and totalizer channels, there is no valid behavior. Calling on a specific channel generates an error. If the digital I/O, DAC, or totalizer channel is in the range of channels, then the channel is ignored.</p>
Details	<ul style="list-style-type: none"> • For delay time, see channel.setdelay() (on page 13-72). • For analog backplane relays with channels, see channel.setbackplane() (on page 13-70). • For channels associated with a channel, see channel.getimage() (on page 13-51). • For channels associated with a channel pattern, see channel.pattern.getimage() (on page 13-62). • For channel states (open/close), see channel.getstate() (on page 13-56). • For closed channels, see channel.getclose() (on page 13-46).
Also see	<p>channel.exclusiveclose() (on page 13-41)</p> <p>channel.exclusiveslotclose() (on page 13-43)</p> <p>channel.open() (on page 13-59)</p> <p>dmm.close() (on page 13-123)</p>

channel.close()	
Example	To close Channels 1 to 5 on Slot 1, Channel 3 on Slot 3, and mychans: <pre>channel.close('1001:1005, 3003, mychans')</pre> To close Channel 1 on Slot 2 and analog backplane relay 3 in Bank 1 on Slot 2: <pre>channel.close('2001, 2913')</pre>
channel.connectrule	
Attribute	Indicates the connection rule for closing and opening channels in the system.
Usage	To read the connect rule: <pre>rule = channel.connectrule</pre> To write the connect rule: <pre>channel.connectrule = rule</pre> Set rule to: channel.BREAK_BEFORE_MAKE or 1 to have BBM connections for relays in system channel.MAKE_BEFORE_BREAK or 2 to have MBB connections for relays in system channel.OFF or 0 to not guarantee a connection rule. The system will close relays as it is able to without adhering to a rule.
Remarks	The connection rule describes the order in which switch channels are opened and closed when using <code>channel.exclusiveclose()</code> (on page 13-41), <code>channel.exclusiveslotclose()</code> (on page 13-43), <code>dmm.close()</code> (on page 13-123), and scanning commands like <code>scan.execute()</code> (on page 13-236) and <code>scan.background()</code> (on page 13-232). These commands may both open and close switch channels in a single command. The connection rule dictates the algorithm used by the instrument to order the opening and closing of switches. When the connection rule is set to <code>channel.BREAK_BEFORE_MAKE</code> , the instrument ensures that all switch channels open before any switch channels close. This behavior covers the most common applications and is considered the safest connection rule because the tested device is completely decoupled from the instrument. This is the default behavior. When switch channels are both opened and closed, this command executes not less than the addition of both the open and close settle times of the indicated switch channels. When the connection rule is set to <code>channel.MAKE_BEFORE_BREAK</code> , the instrument ensures that all switch channels close before any switch channels open. This behavior should be applied with caution because it will connect two test devices together for the duration of the switch close settle time. When switch channels are both opened and closed, the command executes not less than the addition of both the open and close settle times of the indicated switch channels.

channel.connectrule	
Remarks, continued	<p>With no connection rule (set to <code>channel.OFF</code>), the instrument attempts to simultaneously open and close switch channels in order to minimize the command execution time. This results in faster performance at the expense of guaranteed switch position. During the operation, multiple switch channels may simultaneously be in the close position. Make sure your device under test can withstand this possible condition. Cold switching is highly recommended. When switch channels are both opened and closed, the command executes not less than the greater of either the open or close settle times of the indicated switch channels.</p> <p>In general, the settle time of single commands which open and close switch channels depend on a number of factors, such as card type and channel numbers. However, the opening and closing of two sequential channels including no others, can be guaranteed as follows:</p> <pre>channel.BREAK_BEFORE_MAKE open settle time + close settle time channel.MAKE_BEFORE_BREAK open settle time + close settle time channel.OFF maximum of open settle time and close settle time</pre> <p>This behavior is also affected by <code>connect.connectsequential</code> (on page 13-40).</p>
Example	<p>Sets the connect rule in the system to <code>channel.BREAK_BEFORE_MAKE</code>:</p> <pre>channel.connectrule = channel.BREAK_BEFORE_MAKE</pre>

channel.connectsequential	
Attribute	Indicates if the connection rule is sequential or not.
Usage	<p>To read the connect sequential value:</p> <pre>sequential = channel.connectsequential</pre> <p>To write the connect sequential value:</p> <pre>channel.connectsequential = sequential</pre> <p>Set sequential to:</p> <p>channel.OFF or 0 to disable sequential connecting</p> <p>channel.ON or 1 to enable sequential connection</p>
Remarks	<p>If sequential connecting is enabled, the list of channels or analog backplane relays close sequentially. This allows for a deterministic time for the command to be executed. For example, if each channel takes 4ms to close, closing 3 channels takes 12ms. However, if <code>connectsequential</code> was OFF, it may take 4, 8, or 12ms depending on if the card can close multiple relays at once.</p> <p>This attribute applies to switch cards like EMR and reed relay cards.</p> <p>Default setting and <code>channel.reset</code> sequential connection is <code>channel.OFF</code>.</p> <p>Changing this attributes settings causes an existing scan list to be rebuilt based on the new setting.</p>

channel.connectsequential	
Example	Set connect sequential to ON: <code>channel.connectsequential = channel.ON</code>
channel.exclusiveclose()	
Function	Closes the specified items so they are exclusively closed.
Usage	<code>channel.exclusiveclose(<ch_list>)</code> ch_list: A string listing the channels and channel patterns to exclusively close.
Remarks	<p>This function manipulates the channels and analog backplane relays for switching aspects so that only those specified by <code>ch_list</code> are closed.</p> <p>Actions associated with this function include:</p> <ul style="list-style-type: none"> • Opens previously closed channels and analog backplane relays if they are no longer being specified for closure, then closes the desired channels and analog backplane relays as indicated by the items in <code>ch_list</code>. • Settling times are incurred before command processing is complete. The function has no effect on how the DMM is configured and does not use analog backplane relays associated with DMM configuration. • You can specify analog backplane relays in the parameter list. <p>For channel patterns, the analog backplane relays that get manipulated (closed or opened) are the ones that were specified when the pattern was created (see <code>channel.pattern.setimage()</code> (on page 13-63) or <code>channel.pattern.snapshot()</code> (on page 13-66)). However, for channels, they are the ones specified with the <code>channel.setbackplane()</code> (on page 13-70) function. Another option for getting analog backplane relays closed by the command is to include them in the <code>ch_list</code> parameter.</p> <p>If the channel list parameter is an empty string or a string of spaces, all channels and analog backplane relays that are closed are opened. Therefore, when channels or backplane relays are closed, sending <code>channel.exclusiveclose('')</code> is equivalent to <code>channel.open(channel.getclose('allslots'))</code>.</p> <p>However, sending the equivalent commands when nothing is closed generates an error because <code>nil</code> (the response of <code>channel.getclose('allslots')</code>) is being sent to the open command.</p> <p>For digital I/O, DAC, and totalizer channels, there is no valid behavior. Calling on a specific channel generates an error. If the digital I/O, DAC, or totalizer channel is in the range of channels, then the channel is ignored.</p>

channel.exclusiveclose()	
Remarks, continued	<p>An error is generated if</p> <ul style="list-style-type: none"> • Syntax error in parameter string. • The parameter string contains 'slotX', where X = 1 to 6, or 'allslots'. • A specified channel or channel pattern is invalid. • Channel number does not exist for slot specified. • Slot is empty. • Channel pattern does not exist. • A forbidden item is specified. • Does not support being closed like a digital I/O channel. • Channel is paired with another bank for a multi-wire application. <p>Once an error is detected, the command stops processing. Channels open or close only if no errors are found and remain unchanged with any parsing or syntax error.</p> <p>This command allows you to bundle the closing of channels with opening because any currently closed channel or analog backplane relay open if not specified for closure in the parameter. It guarantees that only the specified items are closed on the slots specified in the parameters list.</p>
Details	<ul style="list-style-type: none"> • For delay time, see <code>channel.setdelay()</code> (on page 13-72). • For connection options, see <code>channel.connectrule</code> (on page 13-39) and <code>channel.connectsequential</code> (on page 13-40). • For forbidden channels, see <code>channel.clearforbidden()</code> (on page 13-36), <code>channel.getforbidden()</code> (on page 13-50), and <code>channel.setforbidden()</code> (on page 13-73). • For analog backplane relays with channels, see <code>channel.setbackplane()</code> (on page 13-70). • For channels associated with a channel, see <code>channel.getimage()</code> (on page 13-51). • For channels associated with a channel pattern, see <code>channel.pattern.getimage()</code> (on page 13-62). • For channel states (open/close), see <code>channel.getstate()</code> (on page 13-56). • For closed channels, see <code>channel.getclose()</code> (on page 13-46).
Also see	<p>channel.close() (on page 13-37)</p> <p>channel.exclusiveslotclose() (on page 13-43)</p> <p>channel.open() (on page 13-59)</p> <p>dmm.close() (on page 13-123)</p>

channel.exclusiveclose()	
Example	<p>To only have Channel 3 on Slot 3 closed along with its associated analog backplane relay 3 in Bank 1 on Slot 3:</p> <pre>channel.setbackplane('3003','3913') channel.exclusiveclose('3003')</pre> <p>To eliminate the need for <code>channel.setbackplane</code>:</p> <pre>channel.exclusiveclose('3003, 3913')</pre>
channel.exclusiveslotclose()	
Function	Closes the specified items so they are exclusively closed on slots associated with items in parameter list.
Usage	<pre>channel.exclusiveslotclose(<ch_list>)</pre> <p>ch_list: A string listing the channels and channel patterns to exclusively close on a slot basis.</p>
Remarks	<p>This function manipulates the channels and analog backplane relays for switching aspects so only those specified by <code>ch_list</code> are closed on the slots specified on the items in the parameter list.</p> <p>The actions associated with this function include:</p> <ul style="list-style-type: none"> • Previously closed channels and analog backplane relays are opened if they are no longer being specified for closure on the slots specified with the parameter list items. • Channels and analog backplane relays specified by the items in <code>ch_list</code> are closed. • Settling times are incurred before command processing is complete. • This function has no effect on how the DMM is configured. • You can specify analog backplane relays in the parameter list. <p>For example, if Channel 1 is closed on each of the six slots, specifying a channel list parameter of '2002, 4004' with this command opens Channel 1 on Slot 2 and 4 only. Then, Channel 2 on Slot 2 and Channel 4 on Slot 4 close. Channel 1 remains closed on Slots 1, 3, 5, and 6.</p> <p>For channel patterns, the analog backplane relays that get manipulated (closed or opened) are the ones that were specified when the pattern was created (see <code>channel.pattern.setimage()</code> (on page 13-63) or <code>channel.pattern.snapshot()</code> (on page 13-66)). However, for channels, they are the ones specified with the <code>channel.setbackplane()</code> (on page 13-70) function. Another option for getting analog backplane relays closed by the command is to include them in the <code>ch_list</code> parameter.</p> <p>For digital I/O, DAC, and totalizer channels, there is no valid behavior. Calling on a specific channel generates an error. If the digital I/O, DAC, or totalizer channel is in the range of channels, then the channel is ignored.</p>

channel.exclusiveslotclose()	
Remarks, continued	<p>An error is generated if:</p> <ul style="list-style-type: none"> • Syntax error in parameter string exists. • An empty parameter string is specified, or parameter string with just spaces or a channel pattern that emulates an <code>openall</code> scenario exists. • The parameter string contains 'slotX', where X = 1 to 6, or 'allslots'. • A specified channel or channel pattern is invalid. • Channel number does not exist for slot specified. • Slot is empty. • Channel pattern does not exist. • A forbidden item is specified. • Does not support being closed like a digital I/O channel. • Channel is paired with another bank for a multi-wire application. <p>Once an error is detected, the command stops processing. Channels open or close only if no errors are found and remain unchanged with any parsing or syntax error.</p> <p>This command allows you to bundle the closing of channels with opening because any currently closed channel or analog backplane relay opens if not specified for closure in the parameter. It guarantees that only the specified items are closed on the slots included in parameters list.</p>
Details	<ul style="list-style-type: none"> • For delay time, see <code>channel.setdelay()</code> (on page 13-72). • For connection options, see <code>channel.connectrule</code> (on page 13-39) and <code>channel.connectsequential</code> (on page 13-40). • For forbidden channels, see <code>channel.clearforbidden</code> (on page 13-36), <code>channel.getforbidden()</code> (on page 13-50), and <code>channel.setforbidden()</code> (on page 13-73). • For analog backplane relays with channels, see <code>channel.setbackplane()</code> (on page 13-70). • For channels associated with a channel, see <code>channel.getimage()</code> (on page 13-51). • For channels associated with a channel pattern, see <code>channel.pattern.getimage()</code> (on page 13-62). • For channel states (open/close), see <code>channel.getstate()</code> (on page 13-56). • For closed channels, see <code>channel.getclose()</code> (on page 13-46).
Also see	<p>channel.close() (on page 13-37)</p> <p>channel.exclusiveclose() (on page 13-41)</p> <p>channel.open() (on page 13-59)</p> <p>dmm.close() (on page 13-123)</p>

channel.exclusiveslotclose()	
Example	<p>To have Channel 3 only closed on Slot 3 without affecting any other slot: <code>channel.exclusiveslotclose('3003')</code></p> <p>To only have Channel 5 closed on Slots 1 and 2 without affecting any other slot: <code>channel.exclusiveslotclose('1005, 2005')</code></p> <p>To only open channels on slots of channels in channel pattern MyRoute: <code>channel.exclusiveslotclose('MyRoute')</code></p>

channel.getbackplane()	
Function	Returns a string listing the analog backplane relays controlled when the specified channels are used with commands in a switching aspect.
Usage	<pre>abuslist = channel.getbackplane(<ch_list>)</pre> <p>ch_list: A string listing the channels being queried.</p> <p>abuslist: A string listing analog backplane relays associated with items in <code>ch_list</code>.</p>
Remarks	<p>The response indicates the analog backplane relays that are used or affected by:</p> <ul style="list-style-type: none"> • <code>channel.close()</code> (on page 13-37), used during processing of command. • <code>channel.exclusiveclose()</code> (on page 13-41), used during processing of command. • <code>channel.open()</code> (on page 13-59), used during processing of command. • <code>channel.setbackplane()</code> (on page 13-70) replaces the analog backplane relays with those specified. • <code>channel.setpole()</code> (on page 13-79) clears the analog backplane relays. • <code>scan.execute()</code> (on page 13-236) or <code>scan.background()</code> (on page 13-232), used if channel is configured for switching. • The analog backplane relays indicated by this response are not used or affected by <code>dmm.close()</code> (on page 13-123) or <code>dmm.open()</code> (on page 13-154). • <code>scan.execute()</code> (on page 13-236) or <code>scan.background()</code> (on page 13-232) are not used if channel is configured for measuring. • The parameter string can contain 'slotX', where X equals 1 to 6, or 'allslots'.

channel.getbackplane()	
Remarks, continued	<p>An error is generated if:</p> <ul style="list-style-type: none"> • An empty parameter string is specified. • An empty slot is specified. • A specified channel does not exist for the card installed in a slot. • A channel pattern is specified in parameter list. • A syntax error exists in the parameter list. • A specified channel does not have analog backplane relays associated with it like digital I/O. • An analog backplane relay is specified in parameter list. <p>When <code>ch_list</code> contains multiple items, the string returned has the analog backplane relay channels of a single channel separated by a comma. A semicolon is used to delineate channels.</p> <p>For channel patterns, the analog backplane relays must be specified when creating the pattern in the channel list parameter (see <code>channel.pattern.setimage()</code> (on page 13-63) or <code>channel.pattern.snapshot()</code> (on page 13-66)). Therefore, to see the channels and analog backplane relays associated with a channel pattern, use the <code>channel.pattern.getimage()</code> (on page 13-62) function.</p> <p>Command processing stops as soon as an error is detected and a <code>nil</code> response is then returned. No partial list is returned.</p> <p>For digital I/O, DAC, and totalizer channels, nothing is returned.</p>
Also see	<p>channel.close() (on page 13-37)</p> <p>channel.exclusiveclose() (on page 13-41)</p> <p>channel.open() (on page 13-59)</p> <p>channel.setbackplane() (on page 13-70)</p> <p>channel.setpole() (on page 13-79)</p> <p>Query commands (on page 13-6)</p>
Example	<p>To query analog backplane relay(s) specified on Channel 2 of Slot 2 for switching aspects:</p> <pre>abuslist = channel.getbackplane("2002")</pre>
channel.getclose()	
Function	Queries for the closed channels indicated by the scope of the channel list parameter.
Usage	<pre>closed = channel.getclose(<ch_list>)</pre> <p>ch_list: A string representing the scope of closed items being queried. Items can include channels, backplane relays, and channel patterns.</p> <p>closed: A string listing the channels that are currently closed.</p>

channel.getclose()	
Remarks	<p>The returned string lists the closed channels. If more than one channel is closed, they are comma delimited in the string. If <code>ch_list</code> equals <code>'slotX'</code> where X is 1 to 6, the response indicates the channels that are closed on that specific slot, along with any backplane relays. Likewise, if <code>ch_list</code> equals <code>'allslots'</code>, the response indicates all channels and analog backplane relays that are closed within the system. For channels, the format is <code>SCCC</code> (MUX channels) or <code>SRCC</code> (matrix channels). When the channel list contains a channel pattern, only the channels in that image which are closed are returned.</p> <p>If a single slot is being queried and that slot is empty or does not support the close channel concept, the response is an empty string, meaning no channels are closed on that slot. This allows you to use <code>'allslots'</code> to query for all channels closed and not worry about an error if one of the slots is empty or does not support close channels.</p> <p>The <code>ch_list</code> parameter indicates the scope of channels affected and can include:</p> <ul style="list-style-type: none"> • <code>allslots</code> or <code>slotX</code> (where X equals 1 to 6). • Channel ranges, individual channels, or channel patterns. • Analog backplane relays. <p>If nothing is closed within the specified scope, a <code>nil</code> response is returned.</p> <p>An error is generated if:</p> <ul style="list-style-type: none"> • There is a syntax error in a parameter string. • An empty parameter string is specified. • A non-existent channel or analog backplane relay is specified. • A non-existent channel pattern is specified. <p>Channels of type DAC, totalizer, and digital I/O are omitted from the list.</p>
Also see	<p>channel.close() (on page 13-37)</p> <p>channel.exclusiveclose() (on page 13-41)</p> <p>channel.getstate() (on page 13-56)</p> <p>channel.open() (on page 13-59)</p> <p>channel functions and attributes (on page 13-24)</p> <p>Query commands (on page 13-6)</p>

channel.getclose()	
Example	<p>To see the channels and analog backplane relays that are closed on Slot 5: <code>ClosedSlot5 = channel.getclose('slot5')</code></p> <p>To see all channels and analog backplane relays that are closed in a system: <code>AllClosed = channel.getclose('allslots')</code></p> <p>To see all channels closed within a pattern called 'mychans': <code>ClosedMyChans = channel.getclose('mychans')</code></p> <p>To see all channels closed from Channel 1 to 20 on Slot 3: <code>ClosedRange = channel.getclose('3001:3020')</code></p> <p>To see Channels 1, 2, 3, 5 and analog backplane relay 1 and 2 in Bank 1 on Slot 3: <code>ClosedOnes = channel.getclose('3001, 3002, 3003, 3005, 3911, 3912')</code></p>
channel.getcount()	
Function	Returns a string with the close counts for specified items.
Usage	<p><code>counts = channel.getcount(<ch_list>)</code></p> <p>ch_list: string listing the items to query. Items can include channels, backplane relays, and channel patterns.</p> <p>counts: comma-delimited string listing the channel close counts.</p>
Remarks	<p>This function returns a comma-delimited string of numbers representing the close counts for each channel specified in parameter list. The count values are returned in the same order as the channels were specified. The close counts for an analog backplane relay might be included in the <code>ch_list</code> parameter.</p> <p>The parameter string can contain 'slotX', where X equals 1 to 6, or 'allslots'. It can also contain a pattern which gets translated into the channels and analog backplane relays. Use the channel pattern get command (see <code>channel.pattern.getimage()</code> (on page 13-62)) with the pattern name to see the channels that the close counts pertain to, along with the channel order.</p> <p>An error is generated if:</p> <ul style="list-style-type: none"> • Syntax error in parameter string exists. • An empty parameter string is specified or a parameter string with just spaces exists. • A specified channel is invalid. • The channel number does not exist for slot specified. • Slot is empty. • Does not have a count closure associated with it.

channel.getcount()	
Remarks, continued	<p>If an error is detected, a <code>nil</code> value is returned. No partial list of close counts is returned.</p> <p>When the channel list parameter for this function is <code>'slotX'</code>, the response first lists the channels starting from lowest to highest. After the channels, backplane relays are listed starting with lowest bank first and increasing to the highest.</p> <p>When the channel list parameter for this function is <code>'allslots'</code>, the response starts with Slot 1 and increases to Slot 6. Each slot is processed completely before going to the next. Therefore, all Slot 1 channels and backplane relays are listed before Slot 2 channels.</p> <p>For digital I/O, DAC, and totalizer channels, zero is always returned.</p>
Also see	Query commands (on page 13-6)
Example	<p>To see the close counts for Channels 1 to 5 on Slot 2:</p> <pre>counts = channel.getcount('2001:2005')</pre> <p>To see the close counts for all channels and analog backplane relays on Slot 3:</p> <pre>counts = channel.getcount('slot3')</pre> <p>To see the close counts for channels and analog backplane relays in channel pattern called <code>'mypath'</code>:</p> <pre>MyPathList = channel.pattern.getimage('mypath') print(MyPathList) print(channel.getcount(MyPathList))</pre> <p>or</p> <pre>print(channel.getcount('mypath'))</pre>
channel.getdelay()	
Function	Queries for the additional delay time for the specified items.
Usage	<pre>stime = channel.getdelay(<ch_list>)</pre> <p>ch_list: A string listing the channels to query for their delay time.</p> <p>stime: Returns a comma-delimited string consisting of the delay times (in seconds) for items specified in <code>ch_list</code>.</p>

channel.getdelay()	
Remarks	<p>The parameter string may contain 'slotX', where X equals 1 to 6, or 'allslots'.</p> <p>An error message will be generated for the following reasons:</p> <ul style="list-style-type: none"> • An empty parameter string is specified. • The specified channel does not exist. • Slot is empty. • The specified channel does not exist for card installed in slot. • Parameter syntax error such as incorrect format for <code>ch_list</code>. • An analog backplane relay is specified. • The specified channels does not support a delay time like digital I/O. • A channel pattern is specified. <p>Command processing will stop as soon as an error is detected and a nil response will be generated. No partial list of delay times will be returned.</p> <p>The delay times will be comma delimited in the same order as the items were specified in the <code>ch_list</code> parameter. A value of 0 indicates that no additional delay time is being incurred before a close or open command completes. A command, after updating the state of channels based on the command, will incur the delay time indicated in the response for a channel before completing. However, the internal settling time needs to elapse before the user delay is incurred. Therefore, open/close command processed – settling time incurred – user delay incurred – command completes.</p>
Also see	<p>channel.setdelay() (on page 13-72)</p> <p>Query commands (on page 13-6)</p>
Example	<p>To query Channels 1 and 3 on Slot 5 for their delay time:</p> <pre>mydelaytime = channel.getdelay('5001, 5003')</pre> <p>To see the delay of the channels comprising a channel pattern call <code>mychans</code>:</p> <pre>PatternChannels = channel.pattern.get('mychans') MyDelayPatternTimes = channel.getdelay(PatternChannels)</pre>
channel.getforbidden()	
Function	Returns a string listing the items contained in the channel list parameter that are forbidden to close.
Usage	<pre>forbid_list = channel.getforbidden(<ch_list>)</pre> <p>ch_list: string listing the items to check if they are forbidden to close. Items can include channels, backplane relays, and channel patterns.</p> <p>forbid_list: Comma-separated string listing channels that are forbidden to close in the scope of the channel list.</p>

channel.getforbidden()	
Remarks	<p>The <code>ch_list</code> parameter indicates the scope of channels to check and may include:</p> <ul style="list-style-type: none"> • 'allslots' or 'slotX' (where X equals 1 to 6). • Channel ranges or individual channels. • Analog backplane relays. • Channel pattern(s). <p>If there are no channels in the scope of the channel list that are on the forbidden list, the string returned will be empty or nil. The format of the channels will be SCCC or SRCC in the response string.</p>
Also see	<p>channel.clearforbidden() (on page 13-36)</p> <p>channel.setforbidden() (on page 13-73)</p>
Example	<p>To query for the channels that are "forbidden to close" channels in the system:</p> <pre>MyForbidden = channel.getforbidden('allslots')</pre> <p>To query for channels only on slot3:</p> <pre>MyForbidden = channel.getforbidden('slot3')</pre> <p>To query for channels in specified list:</p> <pre>MyForbidden = channel.getforbidden('1911:1916,2004,2008,2012')</pre>

channel.getimage()	
Function	Query a channel for items associated with that channel when used with a switching aspect command.
Usage	<pre>channels = channel.getimage(<ch_list>)</pre> <p>ch_list: A string representing the channels to query.</p> <p>channels: A string listing the channels and analog backplane relays associated with the specified item(s).</p>
Remarks	<p>An error is generated if:</p> <ul style="list-style-type: none"> • A channel pattern is specified • The specified channel is invalid. • Channel does not exist on the specified slot. • An empty parameter string is specified. <p>The parameter string can contain 'slotX', where X equals 1 to 6, or 'allslots'.</p> <p>The returned string lists the channels in the SCCC or SRCC format. A request for multiple channels is delimited by a semicolon. However, commas delineate the specific channels and analog backplane relays for an individual channel in the string.</p> <p>If an error is detected or the slot is empty, the response is nil.</p> <p>For digital I/O, DAC, and totalizer channels, the channel number itself is returned with no backplanes, pairings, and so on.</p>

channel.getimage()	
Details	To just query for the analog backplane relays associated with a channel, use <code>channel.getbackplane()</code> (on page 13-45).
Also see	channel.pattern.getimage() (on page 13-62) Query commands (on page 13-6) channel functions and attributes (on page 13-24)
Example	<p>Assume Channel 3 on Slot 2 is configured for a 4-wire application and Channel 5 on Slot 2 is configured for a 2-wire application on a 50-channel card.</p> <p>To query Channel 5 on Slot 2:</p> <pre>channels = channel.getimage('2005') print(channels) → 2005</pre> <p>To query Channel 3 on Slot 2:</p> <pre>channels = channel.getimage('2003') print(channels) → 2003(2028)</pre> <p>To query for Channels 2003 and 2005 in a single call:</p> <pre>channels = channel.getimage('2003, 2005') print(channels) → 2003(2028);2005</pre> <p>To query Channel 2028:</p> <pre>channels = channels.getimage('2028') print(channels) → nil -- (error - 2028 is paired for 4-wire operation)</pre>
channel.getlabel()	
Function	Queries for the label associated with one or more channels.
Usage	<pre>label = channel.getlabel(<ch_list>)</pre> <p>ch_list: A string listing the channels to query for the label associated with them.</p> <p>label: String listing the comma-delimited labels for items in <code>ch_list</code>.</p>

channel.getlabel()	
Remarks	<p>The parameter <code>ch_list</code> can contain more than one channel. If it does, a comma is used to delineate the labels for the channels. The return string lists the labels in the same order that the channels were specified.</p> <p>The parameter string can contain <code>'slotX'</code>, where X equals 1 to 6, or <code>'allslots'</code>. It can also contain a label. However, if the label exists, it is in the returned response and not the numerical channel number. For example, if Channel 1001 has a label of <code>"start"</code>, then sending:</p> <pre>print(channel.getlabel('start'))</pre> <p>prints <code>'start'</code> and not the numeric of <code>'1001'</code>.</p> <p>An error is generated if:</p> <ul style="list-style-type: none"> • An empty parameter string is specified. • A specified channel does not exist. • Slot is empty. • Channel not on card. • A channel pattern is specified. • The channel does not support a label setting. • An analog backplane relay is specified. <p>Command processing stops as soon as an error is detected and a <code>nil</code> response is then generated. No partial list of labels is returned.</p> <p>Labels are also supported for digital I/O, DAC, and totalizer channels.</p>
Also see	<p>channel.setlabel() (on page 13-74)</p> <p>Query commands (on page 13-6)</p>
Example	<p>To query for the label of Channel 1 on Slot 1:</p> <pre>MyLabel = channel.getlabel("1001")</pre>

channel.getmatch()	
Function	Gets the match value.
Usage	<pre><match_value> = channel.getmatch(<ch_list>)</pre> <p>match_value: Return string listing the comma-delimited states for channels in <code>ch_list</code>.</p> <p>ch_list: String specifying digital I/O or totalizer channels to query, using normal channel list syntax.</p>
Remarks	<p>If a width greater than 1 was specified with the match value, then the value returned contains the additional channel width specified by the width. For example, the value of 65535 with a width of 2 returns 65535. If the width is 1, then 255 is returned.</p> <p>DAC, backplane, and switch channels are not supported. If included in a range, they are ignored; otherwise, an error is generated.</p>
Also see	channel.setmatch() (on page 13-76)

channel.getmatch()	
Example	Query the match value set to digital I/O Channel 1, Slot 1: <pre>print(channel.getmatch("1001"))</pre>

channel.getmatchtype()	
Function	Gets the match type.
Usage	<pre><match_type> = channel.getmatchtype(<ch_list>)</pre> match_type: Return string listing the comma-delimited states for channels in <code>ch_list</code> . ch_list: String specifying digital I/O or totalizer channels to query, using normal channel list syntax.
Remarks	DAC, backplane, and switch channels are not supported. If included in a range, they are ignored; otherwise, an error is generated.
Also see	channel.setmatchtype() (on page 13-76)
Example	Query the match type for digital I/O Channel 1, Slot 1: <pre>print(channel.getmatchtype("1001"))</pre>

channel.getmode()	
Function	Gets the current mode attribute for a channel.
Usage	<pre><mode> = channel.getmode(<ch_list>)</pre> mode: Return string of a comma-delimited list of modes. ch_list: String specifying digital I/O, DAC, or totalizer channels to query, using normal channel list syntax.
Remarks	Switch and backplane channels do not have modes. If included in a range, they are ignored; otherwise, an error is generated.
Also see	channel.setmode() (on page 13-77)
Example	Query the configuration of the first totalizer channel (Channel 6) in Slot 1: <pre>print(channel.getmode("1006"))</pre>

channel.getoutputenable()	
Function	Gets the current output enable attribute for a channel.
Usage	<pre><relay_state> = channel.getoutputenable(<ch_list>)</pre> state: Return string of a comma-delimited list of relay states. ch_list: String specifying DAC channels to query, using normal channel list syntax.
Remarks	Switch and backplane channels do not have modes. If included in a range, they are ignored; otherwise, an error is generated.
Also see	channel.setoutputenable() (on page 13-78)

channel.getoutputenable()	
Example	Query the state of the first DAC channel on a card in Slot 1: <pre>print(channel.getoutputenable("1010"))</pre>
channel.getpole()	
Function	Queries the pole settings for the specified channels.
Usage	<pre>poles = channel.getpole(<ch_list>)</pre> ch_list: A string listing the channels to query for their pole setting. poles: Returns a string consisting of the poles, comma separated, for ch_list .
Remarks	<p>The parameter string can contain 'slotX', where X equals 1 to 6, or 'allslots'.</p> <p>An error message is generated if:</p> <ul style="list-style-type: none"> • An empty parameter string is specified. • The specified channel does not exist for card installed in slot. • Parameter syntax error such as incorrect format for ch_list. • A channel pattern was specified. • An analog backplane relay was specified. • Channel does not support pole setting like a digital I/O. <p>Command processing stops as soon as an error is detected. No partial list is returned. The response is the numerical value representing the pole selection and not the text. For example, 4-pole selection is 4 and not <code>channel.POLES_FOUR</code>.</p> <p>For channels, manipulate the analog backplane relay channels for the desired pole setting (see <code>channel.setbackplane()</code> (on page 13-70)). Recall channel patterns do not have a pole setting associated with them and have their analog backplane relay channels specified when created (see <code>channel.pattern.getimage()</code> (on page 13-62) and <code>channel.pattern.snapshot()</code> (on page 13-66)).</p> <p>When the channel list parameter for this function is 'slotX', the response first lists the channels starting from lowest to highest.</p> <p>When the channel list parameter for this function is 'allslots', the response starts with Slot 1 and increases to Slot 6. Each slot is processed completely before going to the next. Keeping this in mind, all Slot 1 channels are listed before Slot 2 channels.</p> <p>If an error is detected or the slot is empty, the response is <code>nil</code>.</p> <p>Digital I/O, DAC, backplane, and totalizer channels are not supported.</p>
Also see	channel.setpole() (on page 13-79) Query commands (on page 13-6)
Example	To query Channels 1 and 3 on Slot 5 for pole setting: <pre>mypoies = channel.getpole('5001, 5003')</pre>

channel.getpowerstate()	
Function	Gets the current power state attribute for a channel.
Usage	<pre><state(s)> = channel.getpowerstate (<ch_list>)</pre> <p>state: Return string of a comma-delimited list of power states.</p> <p>ch_list: String specifying the channels to query, using normal channel list syntax.</p>
Remarks	<p>See card-specific documentation for important potential implications (warmup times, effective coverage, use cases, and so on) when disabling power to a channel.</p> <p>Not all channels can be disabled. If included in a range, they are ignored; otherwise, an error is generated.</p>
Also see	channel.setpowerstate() (on page 13-80)
Example	<p>To get the current power state attribute for a channel:</p> <pre>print(channel.getpowerstate ("1006"))</pre>
channel.getstate()	
Function	Queries the state indicators of a channel.
Usage	<pre><state> = channel.getstate(<ch_list>, [<indicator_mask>])</pre> <p>state: Return string listing the comma-delimited states for channels in <code>ch_list</code>.</p> <p>ch_list: String specifying the channels to query, using normal channel list syntax.</p> <p>indicator_mask: Value to specify only certain indicators. If omitted, all indicators are returned.</p>

channel.getstate()	
Remarks	<p>Use this command to query for the state of channels in the system. Each bit in the state represents a different indicator. Therefore, multiple indicators can be present (these values are OR'ed bitwise). Any state or state latch commands behave in this manner.</p> <p>Different channel types support different state information (indicators). The optional state indicator mask can be used to only return certain indicators. If there is no indicator mask, then all indicators are returned.</p> <p>The following status indicators are defined:</p> <ul style="list-style-type: none"> • <code>channel.IND_CLOSED</code> • <code>channel.IND_OVERLOAD</code> • <code>channel.IND_MATCH</code> • <code>channel.IND_OVERFLOW</code> <p>Indicators can be latched or unlatched depending on other system settings. Latched indicators indicate that the condition has occurred since the last reset (or power cycle). Unlatched indicators indicate that the condition has occurred when the <code>channel.getstate()</code> command was issued. The Overflow and Overload indicators default to latched.</p> <p>For switch channels, the only state information is an indicator of relay state (<code>channel.IND_CLOSED</code>).</p> <p>For digital I/O channels, the state information includes an indicator for the state of auto protection and whether the match value has been matched (<code>channel.IND_OVERLOAD</code> and <code>channel.IND_MATCH</code>).</p> <p>For totalizer channels, the state information includes an indicator for overflow and whether the match value has been matched (<code>channel.IND_OVERFLOW</code> and <code>channel.IND_MATCH</code>).</p> <p>For DAC channels, the state information includes an indicator for the state of auto protection (<code>channel.IND_OVERLOAD</code>).</p> <p>For more specific information on the overload and overflow indicators, refer to the documentation for the specific card on which the specified channel resides.</p>
Also see	<p>channel.getclose() (on page 13-46)</p> <p>channel.setmatch() (on page 13-76)</p> <p>channel.setstatelatch() (on page 13-81)</p>

channel.getstate()	
Example	<p>To query the state of the first 20 channels on Slot 4:</p> <pre>MyState = channel.getstate('4001:4020')</pre> <p>To see the state of channels and analog backplane relays in channel pattern called 'mypath':</p> <pre>MyPathList = channel.pattern.getimage('mypath') print(MyPathList) print(channel.getstate(MyPathList))</pre> <p>or</p> <pre>MyPathState = channel.getstate('mypath')</pre> <p>Although the <code>channel.getstate()</code> command returns a string representing a number, this can be easily changed to a number and then compared to one of the provide Lua constants. For example, use the following command to check for an overload on a DAC channel:</p> <pre>TSP> if bit.band(channel.IND_OVERLOAD, tonumber(channel.getstate("4009"))) == 1 then print("OVERLOAD") end</pre> <p>In the previous example, <code>channel.getstate()</code> returns a string that is converted to a number using the Lua <code>tonumber()</code> command. <code>channel.IND_OVERLOAD</code> equates to the number 2. Because the state is a bit-oriented value, the state must be ANDed to the overload constant to isolate it from other indicators.</p> <p>The <code>tonumber()</code> command only works with a single channel. When multiple channels are returned (for example, <code>channel.getstate("slot4")</code>), this string must be parsed by the comma delimiter to find each value.</p>

channel.getstatelatch()	
Function	Gets the mask representing the states which would be latched if they occurred.
Usage	<code>channel.getstatelatch(<ch_list>)</code> ch_list: String specifying the channels to query, using normal channel list syntax.
Also see	channel.setstatelatch() (on page 13-81)
Example	To query the state event latch on digital I/O Channel 1: <code>channel.getstatelatch("1001")</code>

channel.gettype()	
Function	Returns the type associated with a channel.
Usage	<code><type> = channel.gettype(<ch_list>)</code> type: Return string listing the comma-delimited states for channels in <code>ch_list</code> . ch_list: String specifying the channels to query, using normal channel list syntax.

channel.gettype()	
Remarks	<p>The channel type is defined by the physical hardware of the card where the channel resides. The following are valid channel types:</p> <ul style="list-style-type: none"> • <code>channel.TYPE_SWITCH</code> • <code>channel.TYPE_BACKPLANE</code> • <code>channel.TYPE_DAC</code> • <code>channel.TYPE_DIGITAL</code> • <code>channel.TYPE_TOTALIZER</code> <p>Consult the card-specific documentation for more information on the channel types available for a given card.</p>
Example	<p>Query the channel type of Channel 1 in Slot 1:</p> <pre>print(channel.gettype("1001"))</pre>
channel.open()	
Function	Opens specified items in channel list parameter.
Usage	<pre>channel.open(<ch_list>)</pre> <p>ch_list: string listing the items to open. Items can include channels, backplane relays, and channel patterns.</p>

channel.open()	
Remarks	<p>This function opens the specified channels for switching aspects. The items specified in <code>ch_list</code> can include analog backplane relays. For the items specified to open, the channels associated with them open along with the associated analog backplane relays for each. For channel patterns, the analog backplane relays that get opened are the ones that were specified when the pattern was created (see <code>channel.pattern.setimage()</code> (on page 13-63) and <code>channel.pattern.snapshot()</code> (on page 13-66)). However, for channels, they are the ones specified with the <code>channel.setbackplane()</code> (on page 13-70) function. Another option for getting analog backplane relays to be opened by this command is to include them in the <code>ch_list</code> parameter.</p> <p>This command has no effect on how the DMM is configured.</p> <p>To open all channels on a specific slot, use <code>'slotX'</code>, where X = 1 to 6 in the parameter string. To open all channels on all slots, use <code>'allslots'</code> in the parameter string. Using <code>'allslots'</code> has no effect on empty slots or slots that do not support the open command. The <code>'allslots'</code> only applies to slots with channels that support being opened and ignores the ones that do not. This allows you to use <code>'allslots'</code> to open all channels and not worry about an error if one of the slots is empty or does not support open channels.</p> <p>The settling time and user delay associated with a channel needs to elapse before the command completes (see <code>channel.getdelay()</code> (on page 13-49)).</p> <p>The parameter string can contain <code>'slotX'</code>, where X equals 1 to 6, or <code>'allslots'</code>.</p> <p>An error is generated if:</p> <ul style="list-style-type: none"> • Syntax error in parameter string. • An empty parameter string is specified. • A specified channel or channel pattern is invalid • Channel number does not exist for slot specified. • Slot is empty. • Channel pattern does not exist. • Does not support being opened like a digital I/O channel. • Channel is paired with another bank for a multi-wire application. • The <code>slotX</code> specified does not support open like a digital I/O card. <p>Once a parsing error is detected, the command stops processing and no channels are opened. Channels open only if no syntax errors exist in parameter and all channels are valid for opening.</p> <p>For digital I/O, DAC, and totalizer channels, there is no valid behavior. Calling on a specific channel generates an error. If the digital I/O, DAC, or totalizer channel is in the range of channels, then the channel is ignored.</p>

channel.open()	
Details	<p>For delay time, see channel.setdelay() (on page 13-72)</p> <p>For forbidden channels, see channel.setforbidden() (on page 13-73)</p> <p>For analog backplane relays with channels, see channel.setbackplane() (on page 13-70)</p> <p>For channels associated with a channel, see channel.getimage() (on page 13-51)</p> <p>For channels associated with a channel pattern, see channel.pattern.getimage() (on page 13-62)</p> <p>For channel states (open/close), see channel.getstate() (on page 13-56)</p> <p>For closed channels, see channel.getclose() (on page 13-46)</p>
Also see	<p>channel.exclusiveclose() (on page 13-41)</p> <p>dmm.close() (on page 13-123)</p> <p>dmm.open() (on page 13-154)</p> <p>channel.exclusiveslotclose() (on page 13-43)</p> <p>channel.close() (on page 13-37)</p>
Example	<p>To open Channels 1 to 5 on Slot 1, Channel 3 on Slot 3, and mychans:</p> <pre>channel.open('1001:1005, 3003, mychans')</pre> <p>To open all channels on Slot 3 and 5:</p> <pre>channel.open('slot3, slot5')</pre> <p>To open all channels on all slots:</p> <pre>channel.open('allslots')</pre>

channel.pattern.catalog()	
Function	Creates an iterator for the user created channel patterns.
Usage	<pre>for name in channel.pattern.catalog() do ... end</pre>
Remarks	Accessing the catalog for user channel patterns allows the user to print or delete all patterns in volatile memory. The entries will be enumerated in no particular order. This will only list user created channel patterns. It does not list channels which are created at power up.
Also see	<p>channel.pattern.setimage() (on page 13-63)</p> <p>channel.pattern.delete() (on page 13-62)</p> <p>channel.pattern.getimage() (on page 13-62)</p> <p>channel.pattern.snapshot() (on page 13-66)</p>

channel.pattern.catalog()	
Example	<p>To delete all user created channel patterns:</p> <pre>for name in channel.pattern.catalog() do channel.pattern.delete(name) end</pre> <p>To print all user created channel patterns:</p> <pre>for name in channel.pattern.catalog() do print(name) end</pre> <p>To print the names and items associated with all user created channel patterns:</p> <pre>for name in channel.pattern.catalog() do print(name .. " = " .. channel.pattern.getimage(name)) end</pre>
channel.pattern.delete()	
Function	Deletes a channel pattern.
Usage	<pre>channel.pattern.delete(name)</pre> <p>name: A string representing the name of the channel pattern to delete.</p>
Remarks	An error will be generated if the name does not exist.
Also see	<p>channel.pattern.catalog() (on page 13-61)</p> <p>channel.pattern.getimage() (on page 13-62)</p> <p>channel.pattern.setimage() (on page 13-63)</p> <p>channel.pattern.snapshot() (on page 13-66)</p>
Example	<p>To delete a channel pattern called mychans:</p> <pre>channel.pattern.delete("mychans")</pre>
channel.pattern.getimage()	
Function	Query a channel pattern for associated channels and analog backplane relays.
Usage	<pre>channels = channel.pattern.getimage(name)</pre> <p>name: A string representing the name of the channel pattern to query.</p> <p>channels: A string listing channels & analog backplane relays represented by name.</p>

channel.pattern.getimage()	
Remarks	<p>A nil response will be generated if:</p> <ul style="list-style-type: none"> • The channel pattern does not exist. • A channel is specified. • An analog backplane relay is specified. <p>The returned string lists the channels in the SCCC or SRCC format (even if a channel pattern was used to create it). Requests for multiple channel patterns will be delimited by a semicolon, however, commas will delineate the specific channels for a single channel pattern in the string.</p>
Also see	<p>channel.pattern.catalog() (on page 13-61)</p> <p>channel.pattern.delete() (on page 13-62)</p> <p>channel.pattern.snapshot() (on page 13-66)</p> <p>channel functions and attributes (on page 13-24)</p>
Example	<p>Assume <code>mychans</code> is comprised of Channels 1 through 5 on Slot 4 and <code>myroute</code> is comprised of Channels 1, 3, and 5 on Slot 2:</p> <p>To query the channel pattern called <code>mychans</code>:</p> <pre>channels = channel.pattern.getimage('mychans') print(channels) → 4001,4002,4003,4004,4005</pre> <p>To query the channel pattern called <code>myroute</code>:</p> <pre>channels = channel.pattern.getimage('myroute') print(channels) → 2001,2003,2005</pre> <p>To query channel patterns called <code>myroute</code> and <code>mychans</code> in a single call:</p> <pre>channels = channel.pattern.getimage('myroute, mychans') print(channels) → 2001,2003,2005;4001, 4002,4003,4004,4005</pre>

channel.pattern.setimage()	
Function	Creates a channel pattern and associates it with the specified name.
Usage	<pre>channel.pattern.setimage(<ch_list>, name)</pre> <p>ch_list: A string listing the channels, channel patterns, or analog backplane relays to use when creating the new channel pattern.</p> <p>Name: A string representing the name to associate with the new channel pattern.</p>

channel.pattern.setimage()	
Remarks	<p>If the name specified is being used for an existing channel pattern, that pattern is overwritten with the new pattern channel image if no errors occur. The previous image associated with the name is lost. The DMM configuration associated with the pattern remains unchanged in this scenario.</p> <p>An error is generated if</p> <ul style="list-style-type: none"> • The name parameter already exists as a label. • An invalid channel is specified in the channel list parameter. • Slot is empty. • Channel does not exist on slot specified. • Channel is forbidden to close. • A non-existent channel pattern is specified in channel list parameter. • A syntax error exists in either parameter. • Insufficient memory exists to create the channel pattern. • The parameter string contains 'slotX', where X equals 1 to 6, or 'allslots'. • The name parameter contains a space character. • Pattern name exceeds 20 characters. <p>The channel pattern is not created if an error is detected. You can create a channel pattern with an empty <code>ch_list</code> parameter (this would be equivalent to an <code>open all</code>).</p> <p>A channel pattern must include the analog backplane relays as well as desired channels. Once a channel pattern is created, the only way to add a channel or analog backplane relay to an existing pattern is to delete the old and recreate with the new desired items. Or, include the additional channel(s) or analog backplane relay(s) in the channel parameter list with the channel pattern when using.</p> <p>Issuing this function on an existing pattern invalidates the existing scan list (the pattern might or might not be used in the current scan list). Creating a new pattern is okay.</p> <p>Including any channels of type digital I/O, DAC, and totalizer generates an error.</p>

channel.pattern.setimage()	
Details	<p>Channel patterns are not persistent through a power cycle and are stored when a <code>setup.save()</code> (on page 13-254) command is executed. Use <code>setup.recall()</code> (on page 13-253) to restore them.</p> <p>The following restrictions exist when naming a channel pattern:</p> <ul style="list-style-type: none"> • The name must contain only letters, numbers, or underscore. • The name must start with a letter. • The name is case sensitive. <p>Examples of valid names:</p> <p><code>mychans</code>, <code>MyChans</code>, <code>Mychans</code> – three different channel patterns, not one (names are case sensitive)</p> <p><code>Path1</code></p> <p><code>Path20</code></p> <p><code>my_chans</code></p> <p><code>path_3</code></p> <p>Examples of invalid names:</p> <p><code>1path</code> – invalid due to starting with a number</p> <p><code>my chans</code> – invalid due to space</p> <p><code>My, chans</code> – invalid due to comma</p> <p><code>Path1:10</code> – invalid due to colon</p>
Also see	<p>channel.pattern.catalog() (on page 13-61)</p> <p>channel.pattern.delete() (on page 13-62)</p> <p>channel.pattern.getimage() (on page 13-62)</p> <p>channel.pattern.snapshot() (on page 13-66)</p>

channel.pattern.setimage()	
Example	<p>To create a channel pattern called <code>mychans</code> using Channels 1 to 10 on Slot 3:</p> <pre>channel.pattern.setimage('3001:3010', 'mychans')</pre> <p>To add analog backplane relay 1 of Bank 1 on Slot 3 to <code>mychans</code>:</p> <pre>OldList = channel.pattern.getimage('mychans') NewList = OldList .. ',3911' channel.pattern.delete('mychans') channel.pattern.setimage(NewList, 'mychans')</pre> <p>To include 3911 without deleting and creating again:</p> <pre>-- closes mychans and 3911 channel.close('mychans, 3911')</pre> <p>The above works for channels as well. Just replace the <code>', 3911'</code> with the appropriate channel(s). For example, to add Channels 11 and 12 of Slot 3 to <code>mychans</code>.</p> <pre>OldList = channel.pattern.getimage('mychans') NewList = OldList .. ',3011, 3012' channel.pattern.delete('mychans') channel.pattern.setimage(NewList, 'mychans')</pre> <p>To include 3011 and 3012 without deleting and creating again:</p> <pre>channel.close("mychans, 3011, 3012")</pre> <p>To rename <code>MyList</code> to <code>MyPattern</code>:</p> <pre>MyListItems = channel.pattern.getimage('MyList') channel.pattern.setimage(MyListItems, 'MyPattern') channel.pattern.delete('MyList')</pre> <p>or</p> <pre>channel.pattern.setimage(channel.pattern.getimage('MyList'), 'MyPattern') channel.pattern.delete('MyList')</pre>
channel.pattern.snapshot()	
Function	Creates a channel pattern that uses the present state of each channel and analog backplane relay.
Usage	<pre>channel.pattern.snapshot (name)</pre> <p>name: A string representing the name to associate with the present state of channels.</p>

channel.pattern.snapshot()	
Remarks	<p>This command stores the image of closed and opened channels, along with analog backplane relays in the system, and associates them with the name parameter in persistent memory.</p> <p>If the name specified is being used for an existing channel pattern, that pattern is overwritten with the new pattern channel image if no errors occur. The previous image associated with the name is lost. The DMM configuration associated with the pattern remains unchanged in this scenario.</p> <p>An error is generated if:</p> <ul style="list-style-type: none"> • The name parameter already exists as a label. • Insufficient memory exists to save the channel pattern and name in persistent memory. • Pattern name exceeds 20 characters or contains a space. <p>Issuing this function on an existing pattern invalidates the existing scan list (the pattern might or might not be used in the current scan list). Creating a new pattern is okay.</p> <p>Channels of type DAC, totalizer, and digital I/O are ignored.</p>
Details	<p>Not persistent through a power cycle. Channel patterns are stored when a <code>setup.save()</code> (on page 13-254) command is executed. Use <code>setup.recall()</code> (on page 13-253) to restore them.</p> <hr/> <p>NOTE See <code>channel.pattern.setimage()</code> (on page 13-63) for valid name examples.</p>
Also see	<p>channel.pattern.catalog() (on page 13-61)</p> <p>channel.pattern.delete() (on page 13-62)</p> <p>channel.pattern.setimage() (on page 13-63)</p>
Example	<p>To take a snapshot of the current state and name it mysnapshot:</p> <pre>channel.pattern.snapshot("mysnapshot")</pre>

channel.read()	
Function	Reads a value from a channel.
Usage	<pre><value> = channel.read(<ch_list>, [<width>], [<rbuffer>])</pre> <p>value: Return string listing the comma-delimited states for channels in <code>ch_list</code>.</p> <p>ch_list: String specifying channels to read using normal channel list syntax.</p> <p>width: Optional value that specifies reading over multiple consecutive channels.</p> <p>rbuffer: Optional reading buffer to store values read from the channel list.</p>

channel.read()	
Remarks	<p>The width parameter is optional and defaults to 1. The reading buffer parameter is optional. A width does not have to be specified in order to specify a reading buffer. However, if both are specified, the width must be first in the argument list.</p> <p>For digital I/O channels, only a width of 1, 2, 3, or 4 is supported. Any information (bits) greater than the specified width is returned as zero. For example, if Channels 1 and 2 are both 255, a reading with a width of 1 returns 255 and a width of 2 returns 65535. Values read from outputs reflect their current setting. If the read channel is in the overload state, the read value is indeterminate.</p> <p>Totalizer and DAC channels do not support a width other than 1 and result in an error if specified.</p> <p>Switch and backplane channels are not supported.</p> <p>For widths greater than 1, the specified channel occupies the least significant byte. For example, reading the value of 0xff00ff00 from Channel 1 with a width of 4 indicates Channel 1 is 0x00, Channel 2 is 0xff, Channel 3 is 0x00, and Channel 4 is 0xff. Reading the value of 0x00000000 from Channel 1 with a width of 1 indicates Channel 1 is 0x00 and other channels are not included.</p> <p>For a channel with a power state of OFF, the returned value is DISABLED. The value into the reading buffer is indeterminate.</p>
Example	<p>Reading the count from the first totalizer channel (Channel 6) in Slot 1:</p> <pre>count = channel.read("1006")</pre>

channel.reset()	
Function	Resets the channel aspects of the system to factory default settings.
Usage	<pre>channel.reset(<ch_list>)</pre> <p>ch_list: A string list items to reset. Items can include channels, backplane relays, and channel patterns.</p>

channel.reset()	
Remarks	<p>This command resets only the channel aspects for the items specified to factory default settings. For the items specified in the parameter list (<code>ch_list</code>) the following actions occur:</p> <ul style="list-style-type: none"> • For closed channels or analog backplane relays, they open. • For channels, the poles reset to 2 and paired channels are changed to match. • Additional user delay is set to 0. • Labels go back to default of SCCC or SRCC. • Analog backplane relays specified by <code>channel.setbackplane</code> function are cleared. • If channel is forbidden to close, it is cleared from being forbidden to close. • Channels in a channel pattern list are deleted. This means that specifying a channel pattern to reset, resets the items within the pattern and delete that pattern. • Channels have their DMM configurations set to 'nofunction' • The parameter string can contain 'allslots', 'slotX' where X = 1 to 6, channel pattern(s), and channel(s), including a range of channels. • The rest of the settings are unaffected. To reset the entire system to factory default settings, use the <code>reset()</code> (on page 13-230) command. • An error message is generated if the parameter string is empty or just spaces. <p>Using this function to reset a channel or backplane relay involved in scanning invalidates the existing scan list. The list has to be recreated before scanning again.</p> <p>Doing a selective channel reset on some channels can take some time to process, depending on how many patterns exist and how many contain channels being reset. Resetting a channel removes any channel patterns with that channel in its image. Examples of select channel resets:</p> <pre>channel.reset('3003')</pre> <pre>channel.reset('3001:3015')</pre> <pre>channel.reset('slot3')</pre> <p>However, doing a channel reset on for all slots is fast because this needs to remove all patterns since all channels are being reset. For example:</p> <pre>channel.reset('allslots')</pre> <p>or</p> <pre>channel.reset('slot1, slot2, slot3, slot4, slot5, slot6')</pre> <p>For all channels, any trigger settings are removed.</p> <p>For digital I/O channels, the mode is set to input. The match is set to zero (0) and auto-protect is turned on.</p> <p>For totalizer channels, mode is set to falling edge and TTL level.</p> <p>For DAC channels, output is turned off and auto-protect is turned on. Mode is set to -12 to +12 voltage.</p>

channel.reset()	
Also see	dmm.reset() (on page 13-161) reset() (on page 13-230) scan.reset() (on page 13-242) channel functions and attributes (on page 13-24)
Example	<p>To perform a reset on all channels in the system: <code>channel.reset('allslots')</code></p> <p>To reset channels on Slot 1 only: <code>channel.reset('slot1')</code></p> <p>To reset only Channels 1 to 5 on Slot 3: <code>channel.reset('3001:3005')</code></p> <p>To reset only Channel 5 and analog backplane relay 5 in Bank 1 on Slot 5: <code>channel.reset('5005, 5915')</code></p>

channel.resetstatelatch()	
Function	Resets the channel state.
Usage	<code>channel.resetstatelatch(<ch_list>, <state>)</code> ch_list: String specifying the channels to query, using normal channel list syntax. state: String listing the comma-delimited states for channels in <code>ch_list</code> .
Remarks	<p>If the state is reset and the condition still exists, the indicator is reset, but a second event is generated through the channel trigger module.</p> <p>Multiple states can be set by ORing the values together.</p> <p>Use <code>channel.ALL</code> to reset all indicators.</p> <p>Indicators can be latched or unlatched, depending on other system settings. Latched indicators indicate that the condition occurred since the last reset (or power cycle). Unlatched indicators indicate that the condition has occurred when the <code>channel.getstate()</code> (on page 13-56) command was issued. The Overflow and Overload indicators default to latched.</p>
Also see	channel.getstate() (on page 13-56) channel.setstatelatch() (on page 13-81) channel.getstatelatch() (on page 13-58)
Example	<p>To clear out a match indicator that was latched on digital I/O Channel 1: <code>channel.resetstatelatch("1001", channel.IND_MATCH)</code></p>

channel.setbackplane()	
Function	Specify list of analog backplane relays (<code>abuslist</code>) to use with channels specified in <code>ch_list</code> when they are used in a switching aspect.

channel.setbackplane()	
Usage	<p><code>channel.setbackplane(<ch_list>, abuslist)</code></p> <p>ch_list: A string listing the channels to change.</p> <p>abuslist: A string listing analog backplane relays to set for channels in <code>ch_list</code>.</p>
Remarks	<p>The <code>abuslist</code> parameter must specify the entire list of analog backplane relays needed.</p> <p>The analog backplane relays specified in the <code>abuslist</code> parameter are used or affected by:</p> <ul style="list-style-type: none"> • <code>channel.close()</code> (on page 13-37), used during processing of command • <code>channel.exclusiveclose()</code> (on page 13-41), used during processing of command • <code>channel.open()</code> (on page 13-59), used during processing of command • <code>channel.setpole()</code> (on page 13-79) clears the analog backplane relays • <code>scan.execute()</code> (on page 13-236) or <code>scan.background()</code> (on page 13-232), if channel or channel pattern are configured for switching <p>The analog backplane relays specified in the <code>abuslist</code> parameter are not used or affected by:</p> <ul style="list-style-type: none"> • <code>dmm.close()</code> (on page 13-123) • <code>dmm.open()</code> (on page 13-154) • <code>scan.execute()</code> (on page 13-236) or <code>scan.background()</code> (on page 13-232) if channel or channel pattern are configured for measuring <p>The parameter string (<code>ch_list</code>) can contain 'slotX', where X equals 1 to 6, or 'allslots'.</p>

channel.setbackplane()	
Remarks, continued	<p>An error is generated if:</p> <ul style="list-style-type: none"> • An empty slot is specified. • A specified channel or analog backplane relay does not exist for the card installed in a slot. • A syntax error exists in either of the parameters. • An empty parameter string is received for <code>ch_list</code>. An empty string is okay for <code>abuslist</code>. A parameter string of just spaces is treated like an empty string. • A specified channel does not have analog backplane relays associated with it like digital I/O. • An analog backplane relay is specified in <code>ch_list</code>. • A channel is specified in <code>abuslist</code>. • A channel pattern is specified. <p>For channel patterns, the analog backplane relays are specified when the pattern is created (see <code>channel.pattern.setimage()</code> (on page 13-63)).</p> <p>Command processing stops as soon as a parsing or syntax error is detected and no changes are made. Only with no errors are the analog backplane relays updated for the specified channels. When updated, the previous list is replaced with the new specified analog backplane relays in the <code>abuslist</code> parameter.</p> <p>For channels, as their poles setting changes, the list of analog backplane relays gets cleared. Therefore, after changing the poles settings, send this command with the appropriate analog backplane relay channels. (Channel patterns do not have a poles setting associated with them.) When clearing the backplane channels, this can involve clearing the paired channel, whether pairing or un-pairing channels. For example, on a 40-channel card, Channels 1 and 21 are paired when the poles for Channel 1 is set to 4. Therefore, setting the poles setting on Channel 1 to 4 clears the backplane channels for Channels 1 and 21. Likewise, they are both cleared when the poles setting is set back to 2 on Channel 1.</p> <p>Calling this function on an existing channel involved in scanning invalidates the existing scan list.</p> <p>For digital I/O, DAC, and totalizer channels, there is no valid behavior. Calling on a specific channel generates an error. If the digital I/O, DAC, or totalizer channel is in the range of channels, then the channel is ignored.</p>
Also see	<p>channel.close() (on page 13-37)</p> <p>channel.exclusiveclose() (on page 13-41)</p> <p>channel.getbackplane() (on page 13-45)</p> <p>channel.open() (on page 13-59)</p> <p>channel.setpole() (on page 13-79)</p>
Example	<p>To use analog backplane relay 3 and 4 of Slot 2 for switching aspects on Channel 2 of Slot 2:</p> <pre>channel.setbackplane('2002', '2913, 2914')</pre>

channel.setdelay()	
Function	Sets additional delay time for channels specified in <code>ch_list</code> .
Usage	<code>channel.setdelay(<ch_list>, value)</code> ch_list: A string listing channels to modify their delay time. Value: Desired delay time for items in <code>ch_list</code> . Minimum is 0 seconds.
Remarks	<p>An error message will be generated for the following reasons:</p> <ul style="list-style-type: none"> • An empty parameter string is specified. • The value is an invalid setting for the specified channel. • The specified channel doesn't exist for the card installed in the specified slot. • A channel pattern is specified. • The channel is for an empty slot. • The value is invalid for command – parameter out of range error. • Parameter syntax error such as incorrect format for <code>ch_list</code>. • An analog backplane relay is specified. <p>The parameter string may contain 'slotX', where X equals 1 to 6, or 'allslots'.</p> <p>Command processing will stop as soon as an error is detected and no delay times will be modified. Only with no errors, do the specified channels get their delay time changed.</p> <p>Setting the value to 0 indicates to incur no additional delay when closing or opening the specified channels. With a setting of 0, only the needed settling time for a channel to close or open will be incurred. If additional delay is desired then use this command to indicate the amount needed. Channel patterns get their delay from the channels comprising the pattern. Therefore, specify the delay for a pattern through the channels. A pattern will incur the longest delay of all channels comprising that pattern.</p> <p>Calling this function on an existing channel involved in scanning has no affect on the existing scan list.</p>
Details	Setting a delay only applies to switch channels. An error will occur for a read/write channel like digital input/output. The delay being specified by value may be updated based on a card's resolution for delay. To see if the delay value was modified after setting, use the <code>channel.getdelay</code> command to query.
Also see	channel.getdelay() (on page 13-49)
Example	<p>Set Channels 1 and 3 on Slot 5 for a delay time of 50 microseconds:</p> <pre>channel.setdelay("5001, 5003" , 50e-6)</pre> <p>To set the channels on Slot 3 for 0 delay time:</p> <pre>channel.setdelay ("slot3", 0)</pre>

channel.setforbidden()	
Function	Prevents the closing of specified channels.
Usage	<code>channel.setforbidden(<ch_list>)</code> ch_list: A string listing the channels to make forbidden to close.
Remarks	<p>The <code>ch_list</code> parameter indicates the scope of channels affected and may include:</p> <ul style="list-style-type: none"> • <code>allslots</code> or <code>'slotX'</code> (where X equals 1 to 6). • Channel ranges or individual channels • Analog backplane relays. <p>This function prevents all items contained in the channel list parameter from closing (applies the "forbidden to close" attribute to channels specified). To remove the "forbidden to close" attribute, use channel.clearforbidden (on page 13-36).</p> <p>An error will be generated if:</p> <ul style="list-style-type: none"> • The specified channel or analog backplane relay does not exist for card installed in a slot. • The specified channel or analog backplane relay is for an empty slot. • There is a parameter syntax error in the channel specified. • An empty channel list is parsed. <p>Command processing will stop as soon as an error is detected. If an error is found, the forbidden setting is not updated. With no errors, items in the list are marked as being forbidden to close. The previous ones are not lost.</p> <p>After the forbidden list is updated, the existing channel patterns are reviewed. Any of the patterns that contain an item (channel or analog backplane relay) that is now forbidden to close will be deleted. A channel pattern may be specified in the channel list parameter. However, doing this will cause the pattern to be deleted because it contains items that are forbidden to close.</p> <p>Making a channel or backplane forbidden that is involved in scanning invalidates the existing scan list.</p>
Also see	channel.clearforbidden() (on page 13-36) channel.getforbidden() (on page 13-50)
Example	<p>To mark Channels 2, 4, 6, and 8 of Slot 2 as forbidden to close:</p> <pre>channel.setforbidden('2002,2004,2006,2008')</pre> <p>To mark Slot 3 as forbidden to close:</p> <pre>channel.setforbidden('slot3')</pre>
channel.setlabel()	
Function	Sets the label associated with a channel.
Usage	<code>channel.setlabel(<ch_list>, label)</code> ch_list: A string listing the channel to set the label associated with it. Label: A string representing the label for items in <code>ch_list</code> .

channel.setlabel()	
Remarks	<p>This command sets the label of the specified channel in <code>ch_list</code> to the value specified in the label parameter. To clear label use the command with the label parameter equaling an empty string "" or a string with a space as the first character.</p> <p>If the name specified is being used for an existing channel label, that label reverts to the default label and the new channel is updated to use the new label if no errors occur. The previous association with that label is lost. The channel attributes associated with each channel remain unchanged except for their labels. For example, channel one on Slot 4 has a label of 'start'. Sending <code>channel.setlabel('5001', 'start')</code> causes Channel 4001 to lose the label of 'start' and go back to '4001', while Channel 5001's label is set to 'start'. Using 'start' in commands then refers to 5001 and not 4001.</p> <p>An error is generated if:</p> <ul style="list-style-type: none"> • An empty parameter string is specified for <code>ch_list</code>. • Exceeds max length, which is 20 characters. • A specified channel does not exist. • The channel is for an empty slot. • A channel pattern is specified. • The channel does not support a label setting. • An analog backplane relay is specified. • More than one channel is specified in <code>ch_list</code>. • <code>ch_list</code> contains 'slotX' where X = 1 to 6 or 'allslots'. • The label contains a space. However, if the first character is a space, the label is cleared. • The label is already being used to represent a channel pattern. <p>Command processing stops as soon as an error is detected and no channel label is updated. To clear a label back to its factory default, <code>SCCC</code> or <code>SRCC</code>, send an empty string for the label parameter. The label is not persistent through a power-cycle. However, a label is part of data saved with a setup.</p> <p>Labels are also supported for digital I/O, DAC, and totalizer channels.</p>
Details	<ul style="list-style-type: none"> • Not persistent through a power cycle. • You can use labels with commands. Labels and patterns are unique, meaning a channel label cannot be used for a channel pattern. Whichever one is created or specified first is the one used. If you try to use one for the other, an error is generated.
Also see	channel functions and attributes (on page 13-24)

channel.setlabel()	
Example	<p>To set the label for Channel 1 on Slot 1 to "start":</p> <pre>channel.setlabel('1001', 'start')</pre> <p>To clear the label for Channel 1 on Slot 1 back to '1001'</p> <pre>channel.setlabel('1001', '')</pre> <p>or</p> <pre>channel.setlabel('1001', ' ')</pre>

channel.setmatch()	
Function	Sets the match value on a channel.
Usage	<pre>channel.setmatch(<ch_list>, <match value>, [<mask>, <width>])</pre> <p>ch_list: String specifying the channels to query, using normal channel list syntax.</p> <p>match_value: Channel value to compare on the specified channel.</p> <p>mask: Optional value to specify bits used to mask <match_value>.</p> <p>width: Optional value that specifies matches over multiple consecutive channels.</p>
Remarks	<p>The mask is AND'ed bitwise to the match value to determine the final match value used on the channel.</p> <p>The mask and width arguments are optional. A mask must be specified in order to specify a width. The default width is 1. The default mask is <code>channel.ALL</code> (all bits).</p> <p>For digital I/O channels, a width of 1, 2, 3, or 4 channels is supported. Any information (bits) greater than the specified width are ignored. If a width crosses channels, the match status indicator is only on the channel specified in the match value. For example, setting a value with a 2 width on Channel 3 drives the indicator on Channel 3, not Channel 4. Match values for output channels are ignored.</p> <p>Totalizer and DAC channels only support a width of 1.</p> <p>Switch and backplane channels are not supported.</p>
Also see	channel.getmatch() (on page 13-53)
Example	<p>To generate a match state event on Bit 6 of Digital I/O Channel 1:</p> <pre>channel.setmatchtype("1001", channel.MATCH_EXACT) channel.setmatch("1001", 32)</pre>

channel.setmatchtype()	
Function	Sets the match type on a channel.
Usage	<pre>channel.setmatchtype(<ch_list>, <type>)</pre> <p>ch_list: String specifying the channels to set, using normal channel list syntax.</p> <p>type: A value for setting the match operation used on this specific channel.</p>

channel.setmatchtype()	
Remarks	<p>There are four types of match values:</p> <ul style="list-style-type: none"> • <code>channel.MATCH_EXACT</code> • <code>channel.MATCH_ANY</code> • <code>channel.MATCH_NOT_EXACT</code> • <code>channel.MATCH_NONE</code> <p>For an EXACT match, the state match indicator only becomes TRUE when the match value AND match mask value EQUAL the channel read value.</p> <p>For an ANY match, the state match indicator only becomes TRUE when the match value OR match mask value EQUAL the channel read value.</p> <p>For a NOT_EXACT match, the state match indicator only becomes TRUE when the match value AND match mask value AND channel read value are NOT EQUAL to the match value AND match mask value AND last channel read value. In other words, the match value should be the current value, and if the value changes at all away from the original value, then a match is declared.</p> <p>For NONE, matching is effectively disabled. This is the default.</p> <p>For totalizer channels, only MATCH_EXACT and MATCH_NONE are supported. DAC, backplane, and switch channels are not supported.</p>
Also see	channel.getmatchtype() (on page 13-54)
Example	To define the match type for digital I/O Channel 1 to be a MATCH_EXACT type: <code>channel.setmatchtype("1001", channel.MATCH_EXACT)</code>
channel.setmode()	
Function	Sets the mode attribute on a channel.
Usage	<p><code>channel.setmode(<ch_list>, <mode>)</code></p> <p>ch_list: String specifying the channels to set, using normal channel list syntax.</p> <p>mode: The value that sets the mode of a channel's operation.</p>

channel.setmode()	
Remarks	<p>Different channel types contain additional configurable settings. These settings are grouped together by channel type as described in the following paragraphs.</p> <p>For digital I/O channels, the mode indicates the direction of the channel (input or output). The following modes are supported:</p> <ul style="list-style-type: none"> • <code>channel.MODE_INPUT</code> (default) • <code>channel.MODE_OUTPUT</code> • <code>channel.MODE_PROTECT_OUTPUT</code> <p>For totalizer channels, the mode indicates the configuration of the channel (edge and reset). The following modes are supported:</p> <ul style="list-style-type: none"> • <code>channel.MODE_RISING_EDGE</code> • <code>channel.MODE_FALLING_EDGE</code> • <code>channel.MODE_RISING_TTL_EDGE</code> (default) • <code>channel.MODE_FALLING_TTL_EDGE</code> • <code>channel.MODE_RISING_EDGE_READ_RESET</code> • <code>channel.MODE_FALLING_EDGE_READ_RESET</code> • <code>channel.MODE_RISING_TTL_EDGE_READ_RESET</code> • <code>channel.MODE_FALLING_TTL_EDGE_READ_RESET</code> <p>For DAC channels, the mode indicates the output of the channel (function and range). The output is switched off before any mode change is made, and remains off after the mode has changed. The following modes are supported:</p> <ul style="list-style-type: none"> • <code>channel.MODE_VOLTAGE_1</code> • <code>channel.MODE_CURRENT_1</code> • <code>channel.MODE_CURRENT_2</code> • <code>channel.MODE_PROTECT_VOLTAGE_1</code> (default) • <code>channel.MODE_PROTECT_CURRENT_2</code> • <code>channel.MODE_PROTECT_CURRENT_2</code> <p>Changing the mode setting can impact the power consumption of the card. The instrument verifies that power is available before changing the mode. If an insufficient power capability exists, the command generates an error.</p> <p>Consult the card-specific documentation for more detailed information on mode settings and functionality.</p> <p>For digital I/O channels, changing the mode from input to output or from output to input adds an additional channel delay (see <code>channel.setdelay()</code> (on page 13-72)).</p> <p>For switch and backplane channels, there is no valid mode setting. Setting a mode on a specific channel generates an error. If the switch channel is in the range of channels, the mode is ignored.</p> <p>The specified channel list must use only one channel type. For example, channel list "1001:1004" is only valid if Channels 1, 2, 3, and 4 are of the same type. If Channel 3 is a DAC channel, the channel list is invalid and an error is generated.</p>

channel.setoutputenable()	
Function	Sets the output enable attribute on a channel.
Usage	<code>channel.setoutputenable(<ch_list>, <state>)</code> ch_list: String specifying the channels to set, using normal channel list syntax. state: A value representing the desired state of the channel's output.
Remarks	<p>Channels with output OFF consume less power.</p> <p>For DAC channels, output enable is used to indicate the whether or not the DAC is driving the output. The following possible states are supported:</p> <ul style="list-style-type: none"> • channel.ON • channel.OFF (default) <p>For DAC channels, changing the output state adds an additional channel delay (see <code>channel.setdelay()</code> (on page 13-72)).</p> <p>Channels with output OFF consume less power. Changing the output setting impacts the power consumption of the card. The instrument verifies that power is available before changing the mode. If an insufficient power capability exists, the command generates an error. Consult the specific card documentation for more detailed information on a channel's output characteristics.</p> <p>For switch, backplane, digital I/O, and totalizer channels, there is no valid output enable attribute. Setting output enable on a specific channel generates an error. If the switch or totalizer channel is in the range of channels, the mode is ignored.</p>
Also see	channel.getoutputenable() (on page 13-54)
Example	To turn the output off on the first DAC channel (Channel 10) in Slot 1: <code>channel.setoutputenable("1010", channel.OFF)</code>

channel.setpole()	
Function	Specifies the pole setting for a list of channels.
Usage	<code>channel.setpole(<ch_list>, value)</code> ch_list: A string listing the channels to assign their pole setting value: Desired pole setting for items in <code>ch_list</code> . Use the following: <ul style="list-style-type: none"> • For one-pole: <code>channel.POLES_ONE</code> or 1. • For two-pole: <code>channel.POLES_TWO</code> or 2. • For four-pole: <code>channel.POLES_FOUR</code> or 4.

channel.setpole()	
Remarks	<p>An error message is generated for the following reasons:</p> <ul style="list-style-type: none"> • An empty parameter string is specified. • The value parameter is an invalid setting for the specified channel. • The specified channel does not exist for the card installed in a slot. • The channel is for an empty slot. • The value parameter is invalid for command – parameter out of range error. • Parameter syntax error such as incorrect format for <code>ch_list</code>. • A channel pattern or analog backplane relay was specified. <p>The parameter string can contain 'slotX', where X equals 1 to 6, or 'allslots'.</p> <p>Command processing stops as soon as an error is detected and no pole settings are modified. Only with no errors are the poles setting changed on the specified channels.</p> <p>Recall channel patterns do not have a pole setting associated with them. For channel patterns, the analog backplane relays must be specified when creating the pattern (see <code>channel.pattern.setimage()</code> (on page 13-63) and <code>channel.pattern.snapshot()</code> (on page 13-66)).</p> <p>You manipulate the analog backplane relays for the desired pole setting by using the <code>channel.setbackplane()</code> (on page 13-70) function for channels. For channels, as the pole setting changes, their analog backplane relays, specified by <code>channel.setbackplane()</code> (on page 13-70), get cleared. Therefore, after a pole setting change, you need to add the desired analog backplane relays for desired pole setting by using <code>channel.setbackplane()</code> (on page 13-70).</p> <p>The analog backplane relays get manipulated based on the DMM configuration assigned to a channel when the channel used with the <code>dmm.close()</code> (on page 13-123) command. When clearing the backplane channels, this involves clearing the paired channel whether pairing or unpairing channels. For example, on a 40-channel card, Channels 1 and 21 are paired when the poles for Channel 1 is set to 4. Therefore, when changing the poles setting on Channel 1 to 4, the backplane channels for Channels 1 and 21 are cleared. Likewise, they both are cleared when the poles setting is set back to 2 on Channel 1.</p> <p>Calling this function on an existing channel involved in scanning invalidates the existing scan list.</p> <p>For digital I/O, DAC, and totalizer channels, only a value of 1 is accepted.</p>
Also see	<p>channel.getbackplane() (on page 13-45)</p> <p>channel.getpole() (on page 13-55)</p> <p>channel.setbackplane() (on page 13-70)</p>
Example	<p>Set Channels 1 and 3 on Slot 5 to four-pole:</p> <pre>channel.setpole("5001, 5003", channel.POLES_FOUR)</pre>

channel.setpowerstate()	
Function	Sets the power state attribute on a channel.
Usage	<pre>channel.setpowerstate(<ch_list>, <state>)</pre> <p>ch_list: String specifying the channels to query, using normal channel list syntax.</p> <p>state: A value representing the desired channel's power state.</p>
Remarks	<p>Channels with an OFF power state consume less power. When a channel previously OFF is turned ON, the channel attributes are reset to their default values (except the power state attribute).</p> <p>The default value is dependent on the installed card. The following possible values are supported:</p> <ul style="list-style-type: none"> • channel.ON • channel.OFF <p>Changing the output setting impacts the power consumption of the card. The instrument verifies that power is available before changing the mode. If an insufficient power capability exists, the command generates an error.</p> <p>Consult the specific card documentation for more detailed information on a channel's power usage characteristics, including default state, possible warmup issues, and effects on other channels.</p> <p>When a channel with an OFF power state is used in a scan, results are undefined. No error notification is issued.</p> <p>For switch, backplane, and digital I/O channels, there is no valid power state attribute. Setting the power state on a specific channel generates an error.</p> <hr/> <p>NOTE: On some cards for DAC channels, there can be a warmup time for the DAC to reach full accuracy (see card-specific documentation).</p> <p>On some cards for totalizer channels, setting the power state of a single channel can affect the power state of other channels. If a single totalizer channel is turned ON, then all totalizer channels are reset to their defaults.</p>
Also see	channel.getpowerstate() (on page 13-55)
Example	<pre>channel.setpowerstate("1010", channel.ON)</pre>

channel.setstatelatch()	
Function	Sets the state indicators to either latching or non-latching.
Usage	<pre>channel.setstatelatch(<ch_list>, <state latch mask>)</pre> <p>ch_list: String specifying the channels to set, using normal channel list syntax.</p> <p>state latch mask: A value specifying the indicators to latch.</p>

channel.setstatelatch()	
Remarks	<p>Each indicator is represented by a bit in the mask.</p> <p>For non-latching applications, the state indicator clears automatically when the causing condition clears itself. For latching applications, the condition is cleared using the <code>channel.resetstate()</code> command.</p> <p>When using the trigger module, events are always non-latching (or pulse oriented). However, in latching mode, the event is only generated once at the beginning. In non-latching mode, the event is generated anytime the condition begins.</p> <p>Set multiple states by ORing the values together.</p>
Also see	<p>channel.getState() (on page 13-56)</p> <p>channel.getstatelatch() (on page 13-58)</p>
Example	<p>To generate a match state event on digital I/O Channel 1:</p> <pre>channel.setstatelatch("1001", channel.IND_MATCH)</pre>

channel.trigger[N].clear()	
Function	Clears any pending events.
Usage	<code>channel.trigger[N].clear()</code>
Also see	channel.trigger[N].set() (on page 13-82)
Example	<p>To clear any pending events on channel trigger 1:</p> <pre>channel.trigger[1].clear()</pre>

channel.trigger[N].EVENT_ID	
Function	Defined constant that indicates the event ID in the event system.
Usage	<code>channel.trigger[N].EVENT_ID</code>
Also see	channel.trigger[N].set() (on page 13-82)
Example	<p>To use a channel trigger event to start a scan:</p> <pre>scan.trigger.chan.stimulus = channel.trigger[1].EVENT_ID</pre>

channel.trigger[N].get()	
Function	Gets trigger information associated with a given trigger.
Usage	<pre><ch_list>, <state_match> = channel.trigger[<n>].get()</pre> <p>ch_list: Return string specifying the channels watched by this trigger.</p> <p>state_match: Return value specifying the state to match when triggering an event.</p>
Also see	channel.trigger[N].set() (on page 13-82)
Example	<pre>chan_list, state_match = channel.trigger[1].get() print(chan_list, state_match)</pre>

channel.trigger[N].set()	
Function	Sets the channel status trigger module to watch the state of a specific channel.
Usage	<code>channel.trigger[N].set(<ch_list>, <state_match>)</code> ch_list: String specifying the channels to query, using normal channel list syntax. state_match: Value of the state indicators which are to be matched.
Remarks	If the channel list contains more than one channel, then the trigger acts as a logical OR. When any one of the channels in the list matches the desired state, a trigger event is generated. Therefore, if an indicator is present in both the match and the actual state, then an event is triggered. If the match contains more than one state indicator, only one of those indicators needs be present to trigger the event. There are a total of eight channel trigger events per Model 3706, defined by [N]. Using this mechanism, a trigger can be generated when a pattern is matched on an I/O, a totalizer matches a defined count, or an I/O has an over-current condition. Latching functionality is not supported. Switch channels are currently not supported. To clear a trigger that is no longer needed, pass an empty channel list ("").
Also see	channel.trigger[N].get() (on page 13-82)
Example	To define channel trigger event 1 to occur when digital I/O Channel 1 matches its defined match value: <code>channel.trigger[1].set("1001", channel.IND_MATCH)</code>

channel.trigger[N].wait()	
Function	Waits for the desired trigger or timeout period, whichever comes first.
Usage	<code><triggered> = channel.trigger[N].wait(<timeout>)</code> triggered: Returns an indication that a trigger occurred. timeout: Specifies the number of seconds to wait.
Remarks	If one or more trigger events were detected since the last time <code>channel.trigger[N].wait</code> or <code>channel.trigger[N].clear</code> was called, this function returns immediately. After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected. The value for timeout must be greater than zero and less than 10,000.
Example	To wait 5 seconds for channel trigger event 1: <code>channel.trigger[1].wait(5)</code>

channel.write()	
Function	Writes a value to a channel.
Usage	<pre>channel.write(<ch_list>, <value>, [<width>])</pre> <p>ch_list: String specifying channels to write, using normal channel list syntax.</p> <p>value: The value to be written to the channel.</p> <p>width: Optional value that specifies the channel width of the write.</p>
Remarks	<p>For widths greater than 1, the specified channel occupies the least significant byte. For example, writing the value of 0xff00ff00 to Channel 1 with a width of 4 sets Channel 1 to 0x00, Channel 2 to 0xff, Channel 3 to 0x00, and Channel 4 to 0xff. Writing the value of 0xff00ff00 to Channel 1 with a width of 1 sets Channel 1 to 0x00 and leaves other channels untouched.</p> <p>For digital I/O channels, only a width of 1, 2, 3, or 4 is supported. Any other widths are ignored. Values written to inputs are ignored. If no specified channel is set for output, then an error is generated. If a width crosses channels, then only the channels set to output are affected.</p> <p>Totalizers, DACs, and switch channels do not support a width other than 1. Specifying a width greater than 1 results in an error.</p> <p>For a channel with a power state of OFF, an error is generated. No action is taken on any channel in the specified channel list.</p> <p>For DAC channels, the value is expected to be the desired floating point voltage or current. Also, an error is generated if the value is out of range. No action is taken on any channel in the specified channel list.</p> <p>For digital I/O channels, the value becomes the settings of the digital output.</p> <p>For totalizer channels, the value becomes the new current totalizer count.</p> <p>The time it takes to execute the write command is affected by the channel delay setting.</p>
Example	<p>To output a value of 33 to digital I/O Channel 1:</p> <pre>channel.write("1001", 33)</pre>

dataqueue functions and attributes

Use the dataqueue 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 at any time. You can access data from the data queue even if a remote group or a node has overlapped operations in process.

dataqueue.add()	
Function	Store an item of data in the data queue.
Usage	<pre>success = dataqueue.add(value[, timeout])</pre> <p>value: The data item to add.</p> <p>timeout: Maximum amount of time in seconds to wait for room in the queue if it is full.</p> <p>success: Success indication.</p>
Remarks	<p>This function will add one entry to the data queue. If the queue is full, this function will wait up to <code>timeout</code> seconds for room to be made available. This function will return true if the value was added to the data queue. It will return false if the queue is full and the item could not be added before the timeout expires.</p> <p>The timeout value may only be specified when called from the local node. If a timeout value is not given, the function will not wait for room in the queue if it is full.</p> <hr/> <p>NOTE If <code>value</code> is a table, this function will make a deep copy of it rather than storing a reference to it. This will make a complete copy of all entries within the table, including all nested tables.</p>

dataqueue.CAPACITY	
Attribute	The maximum number of entries the data queue can hold.
Usage	<pre>capacity = dataqueue.CAPACITY</pre> <p>capacity: Maximum number of entries in the data queue.</p>
Remarks	This constant indicates the maximum number of values that can be stored in the data queue.

dataqueue.clear()	
Function	Clear the data queue.
Usage	<pre>dataqueue.clear()</pre>
Remarks	This function will remove all entries from the data queue. If any nodes are waiting to add data to the queue, this method will force them to fail as if they timed out.

dataqueue.count	
Attribute	The number of entries currently stored in the data queue.
Usage	<code>count = dataqueue.count</code> count : Number of entries in the data queue.
Remarks	This attribute is a read-only attribute that indicates how many entries are in the data queue.

dataqueue.next()	
Function	Retrieve an entry from the data queue.
Usage	<code>value = dataqueue.next([timeout])</code> timeout : Maximum amount of time in seconds to wait for data if the queue is empty. value : The next entry from the data queue.
Remarks	This function will remove the next entry from the data queue and return its value. If the queue is empty, this function will wait up to <code>timeout</code> seconds for data to arrive. If no data arrives before the timeout expires, this function will return nil. The timeout value may only be specified when called from the local node. If a timeout value is not given, the function will not wait for data to be put in the queue if it is empty. NOTE If the entry is a table, this function will return a deep copy of its contents at the time the table was added to the data queue rather than returning a reference to the original table.

delay functions

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	<code>delay(seconds)</code> seconds : Set delay in seconds (100000 seconds maximum).
Remarks	<ul style="list-style-type: none"> This function will cause a delay for the specified number of seconds. It is impossible to delay for zero seconds. Delays smaller than 50μs (seconds) 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	To pause program execution for 50ms: <code>delay(0.050)</code>

digio functions and attributes

Use the functions and attributes in this group 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	<code>data = digio.readbit(N)</code> N: Digital I/O line number to be read (1 to 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.
Also see	digio.readport() (on page 13-87) digio.writebit() (on page 13-92) digio.writeport() (on page 13-92)
Example	Assume line 4 is set high, and it is then read: <code>data = digio.readbit(4)</code> <code>print(data)</code> Output: 1.000000e+00

digio.readport()	
Function	Reads the digital I/O port.
Usage	<code>data = digio.readport()</code>
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 0000010101010. Lines 2, 4, 6 and 8 are high (1), and the other 10 lines are low (0).
Also see	digio.readbit() (on page 13-87) digio.writebit() (on page 13-92) digio.writeport() (on page 13-92)

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

digio.trigger[N].assert()	
Function	Asserts a trigger on one of the digital I/O lines.
Usage	<code>digio.trigger[N].assert()</code> N : Digital I/O trigger line: 1 to 14
Remarks	The set pulsewidth determines how long the trigger is asserted.
Also see	digio.trigger[N].pulsewidth (on page 13-90)
Example	Asserts trigger on I/O line 2: <pre>digio.trigger[2].assert()</pre>

digio.trigger[N].clear()	
Function	Clears a trigger event on a digital I/O line.
Usage	<code>digio.trigger[N].clear()</code> N : Digital I/O trigger line: 1 to 14
Remarks	A trigger's event detector remembers if a trigger event has been detected since the last <code>digio.trigger[N].wait()</code> call. This function clears a trigger's event detector and discards the previous history of the trigger line.
Also see	digio.trigger[N].wait() (on page 13-92)
Example	Clears trigger event on I/O line 2: <pre>digio.trigger[2].clear()</pre>

digio.trigger[N].mode	
Attribute	Controls the mode in which the trigger event detector as well as the output trigger generator will operate on the given trigger line.

digio.trigger[N].mode	
Usage	<p>To read the trigger mode:</p> <pre>tmode = digio.trigger[N].mode</pre> <p>To write the trigger mode:</p> <pre>digio.trigger[N].mode = tmode</pre> <p>N: Digital I/O trigger line 1 to 14</p> <p>Set tmode to one of the following values:</p> <p>digio.TRIG_BYPASS or 0: Allow direct control of the line.</p> <p>digio.TRIG_FALLING or 1: Detect falling edge triggers as input. Assert a TTL-low pulse for output.</p> <p>digio.TRIG_RISING or 2: Use digio.TRIG_RISINGA if the line is in the high output state. Use digio.TRIG_RISINGM if the line is in the low output state.</p> <p>digio.TRIG_EITHER or 3: Detect rising or falling edge triggers as input. Assert a TTL-low pulse for output.</p> <p>digio.TRIG_SYNCHRONOUS or 4: Detect falling edge triggers as input and automatically latch and drive them low when detected. Release a latched line for output.</p> <p>digio.TRIG_SYNCHRONOUS or 5: Detect falling edge triggers as input and latch them low. Assert a TTL-low pulse for output.</p> <p>digio.TRIG_SYNCHRONOUSM or 6: Detect rising edge triggers as input. Assert a TTL-low pulse for output.</p> <p>digio.TRIG_RISINGA or 7: Detect rising edge triggers as input. Assert a TTL-low pulse for output.</p> <p>digio.TRIG_RISINGM or 8: Assert a TTL-high pulse for output. Input edge detection is not possible in this mode.</p>
Remarks	<p>The default trigger mode for a line will be TRIG_BYPASS. In this mode, the line can be directly controlled as a digital I/O line. When programmed to any other mode, the output state of the I/O line is controlled by the trigger logic and the user specified output state of the line will be ignored.</p> <p>For compatibility with older firmware, when the trigger mode is set to TRIG_RISING, the user specified output state of the line will be examined. If the output state selected when the mode is changed is high, the actual mode used will be TRIG_RISINGA. If the output state selected when the mode is changed is low, the actual mode used will be TRIG_RISINGM.</p> <p>TRIG_SYNCHRONOUS is provided for compatibility with older firmware. Either TRIG_SYNCHRONOUSA or TRIG_SYNCHRONOUSM should be used instead.</p> <ul style="list-style-type: none"> • tmode can be expressed as a number or as one of the pre-defined constants (see the "Usage" section above). • When reading the trigger mode, it is returned as a number with the modes noted, tmode is defined.
Example	<p>To set the trigger mode for I/O line 4 to TRIG_RISING:</p> <pre>digio.trigger[4].mode = 2</pre>

digio.trigger[N].overrun	
Attribute	Event detector overrun status.
Usage	<pre>overrun = digio.trigger[N].overrun</pre> <p>overrun: Trigger overrun state. N: Digital I/O trigger line: 1 to 14</p>
Remarks	This attribute is a read-only attribute that indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the event. It also is not an indication of an output trigger overrun.
Example	<pre>overrun = digio.trigger[1].overrun</pre> <pre>print(overrun) → false</pre>

digio.trigger[N].pulsewidth	
Attribute	The length of time that the trigger line will be asserted for output triggers. N is a digital I/O trigger line: 1 to 14
Usage	<p>To read pulse width:</p> <pre>width = digio.trigger[N].pulsewidth</pre> <p>To write pulse width:</p> <pre>digio.trigger[N].pulsewidth = width</pre> <p>pulsewidth: length in s (seconds). N: Digital I/O trigger line: 1 to 14</p>
Remarks	<ul style="list-style-type: none"> The trigger line is guaranteed to be asserted for at least the specified time, and it might be asserted slightly longer. Setting pulsewidth to 0 (seconds) asserts the trigger indefinitely. The default pulsewidth time is 10µs for digio lines 1 through 9, and 20µs for digio lines 10 through 14
Also see	See digio.trigger[N].release() (on page 13-90).
Example	<p>Sets pulsewidth for trigger line 4 to 20µs:</p> <pre>digio.trigger[4].pulsewidth = 20e-6</pre>

digio.trigger[N].release()	
Function	Releases an indefinite length or latched trigger.
Usage	<pre>digio.trigger[N].release()</pre> <p>N: Digital I/O trigger line: 1 to 14</p>
Remarks	Releases a trigger that was asserted with an indefinite pulsewidth, as well as a trigger that was asserted in response to receiving a synchronous mode trigger. Only the specified trigger line (N) is affected.
Also see	digio.trigger[N].pulsewidth (on page 13-90)

digio.trigger[N].release()	
Example	Releases trigger line 4: <code>digio.trigger[4].release()</code>
digio.trigger[N].stimulus	
Attribute	Event to cause this trigger to assert.
Usage	<code>triggerstimulus = digio.trigger[N].stimulus</code> - or - <code>digio.trigger[N].stimulus = triggerstimulus</code> N: Digital I/O trigger line: 1 to 14 triggerstimulus: The event identifier for the triggering event.
Remarks	This attribute selects which event(s) will cause digital output line to assert a trigger. Set this attribute to 0 to bypass waiting for an event. Trigger stimulus for a digio line may be set to one of the following (existing trigger event IDs): <ul style="list-style-type: none">• <code>digio.trigger[N].EVENT_ID</code>: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.• <code>display.trigger.EVENT_ID</code>: The trigger key on the front panel is pressed.• <code>trigger.EVENT_ID</code>: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event.• <code>trigger.blender[N].EVENT_ID</code>: A combination of events has occurred.• <code>trigger.timer[N].EVENT_ID</code>: A delay expired.• <code>tsplink.trigger[N].EVENT_ID</code>: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line.• <code>lan.trigger[N].EVENT_ID</code>• <code>scan.trigger.EVENT_SCAN_READY</code>: Scan Ready Event.• <code>scan.trigger.EVENT_SCAN_START</code>: Scan Start Event• <code>scan.trigger.EVENT_CHANNEL_READY</code>: Channel Ready Event• <code>scan.trigger.EVENT_MEASURE_COMP</code>: Measure Complete Event• <code>scan.trigger.EVENT_SEQUENCE_COMP</code>: Sequence Complete Event• <code>scan.trigger.EVENT_SCAN_COMP</code>: Scan Complete Event• <code>scan.trigger.EVENT_IDLE</code>: Idle Event <hr/> NOTE Use ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because they may need to change when enhancements are added to the instrument.
Example	To set the trigger stimulus of digital I/O line 3 to be the channel ready event during a scan: <code>digio.trigger[3].stimulus = scan.EVENT_CHANNEL_READY</code> To clear the trigger stimulus of digital I/O line 3: <code>digio.trigger[3].stimulus = 0</code>

digio.trigger[N].wait()	
Function	Waits for a trigger.
Usage	<pre>triggered = digio.trigger[N].wait(timeout)</pre> <p>N: Digital I/O trigger line: 1 to 14</p> <p>timeout: Set timeout in seconds.</p> <p>triggered: Returns `true` if a trigger was detected, or `false` if no triggers were detected during the timeout period.</p>
Remarks	This function will wait up to timeout seconds for an input trigger. If one or more trigger events were detected since the last time <code>digio.trigger[N].wait</code> (this function) or digio.trigger[N].clear() (on page 13-88) was called, this function will return immediately. After waiting for a trigger with this function, the event detector will be automatically reset and re-armed. This is true regardless of the number of events detected.
Also see	digio.trigger[N].clear() (on page 13-88)
Example	<p>Waits up to three seconds for a trigger to be detected on trigger line 4, then displays if the trigger was detected:</p> <pre>triggered = digio.trigger[4].wait(3) print(triggered)</pre> <p>Output: <code>false</code> (no triggers detected) <code>true</code> (trigger detected)</p>

digio.writebit()	
Function	Sets a digital I/O line high or low.
Usage	<pre>digio.writebit(bit, data)</pre> <p>bit: Digital I/O line number (1 to 14)</p> <p>data: Value to write to the bit; 0 (low) or 1 (high)</p>
Remarks	<ul style="list-style-type: none"> If the output line is write protected, using the digio.writeprotect() (on page 13-93) attribute, the command will be ignored. The reset function does not affect the present states of the digital I/O lines. <p>NOTE The second parameter (data) needs to be 0 to clear the bit. Any value other than 0 causes the bit to be set.</p>
Also see	digio.readbit() (on page 13-87) digio.readport() (on page 13-87) digio.writeport() (on page 13-92)
Example	<p>Sets digital I/O line 4 low (0):</p> <pre>digio.writebit(4, 0)</pre>

digio.writeport()	
Function	Writes to all digital I/O lines.
Usage	<pre>digio.writeport(data)</pre> <p>data: Value to write to the port; 0 to 16383.</p>

digio.writeport()	
Remarks	<ul style="list-style-type: none"> 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() (on page 13-93)). The reset function does not affect the present states of the digital I/O lines.
Also see	digio.readbit() (on page 13-87) digio.readport() (on page 13-87) digio.writebit() (on page 13-92)
Example	Sets digital I/O lines 1 through 8 high (binary 00000011111111): <code>digio.writeport(255)</code>

digio.writeprotect	
Attribute	Write protect mask that disables bits from being changed with the digio.writebit() (on page 13-92) and digio.writeport() (on page 13-92) functions.
Usage	<p>To read writeprotect mask:</p> <pre>mask = digio.writeprotect</pre> <p>To write writeprotect mask:</p> <pre>digio.writeprotect = mask</pre> <p>mask: Set to the value that specifies the bit pattern for write protect.</p>
Remarks	<ul style="list-style-type: none"> Bits that are set to one cause the corresponding line to be write protected. The binary equivalent of mask indicates the mask to be set for the I/O port. For example, a mask value of 7 has a binary equivalent 00000000000111. This mask write protects lines 1, 2 and 3.
Example	Write protects lines 1, 2, 3 and 4: <code>digio.writeprotect = 15</code>

display functions and attributes

The functions and attributes in this group are used for various display operations.

display.clear()	
Function	Clears all lines of the display.
Usage	<code>display.clear()</code>

display.clear()	
Remarks	<ul style="list-style-type: none"> • This function will switch to the user screen and then clear the display. • The <code>display.clear()</code>, display.setcursor() (on page 13-105), and display.settext() (on page 13-106) 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.
Also see	display.setcursor() (on page 13-106) display.settext() (on page 13-105)
display.getannunciators()	
Function	Reads the annunciators that are presently turned on.
Usage	<pre>annun = display.getannunciators()</pre> <p>annun: Returns the bitmap value for annunciators that are turned on.</p>

display.getannunciators()																																																											
Remarks	<p>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:</p> <pre>0000010000000100</pre> <p>The above bitmap indicates that bits 3 and 11 are set. From the chart below, bit 3 and bit 11 corresponds to the annunciators that are turned on (4W and REM). Notice that the sum of the weighted values for bits 3 and 11 is the returned value (1028).</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Annunciator</th> <th>Bit</th> <th>Weighted Value</th> <th>Annunciator</th> <th>Bit</th> <th>Weighted Value</th> </tr> </thead> <tbody> <tr> <td>FILT</td> <td>1</td> <td>1</td> <td>EDIT</td> <td>9</td> <td>256</td> </tr> <tr> <td>MATH</td> <td>2</td> <td>2</td> <td>ERR</td> <td>10</td> <td>512</td> </tr> <tr> <td>4W</td> <td>3</td> <td>4</td> <td>REM</td> <td>11</td> <td>1024</td> </tr> <tr> <td>AUTO</td> <td>4</td> <td>8</td> <td>TALK</td> <td>12</td> <td>2048</td> </tr> <tr> <td>ARM</td> <td>5</td> <td>16</td> <td>LSTN</td> <td>13</td> <td>4096</td> </tr> <tr> <td>TRIG</td> <td>6</td> <td>32</td> <td>SRQ</td> <td>14</td> <td>8192</td> </tr> <tr> <td>*(star)</td> <td>7</td> <td>64</td> <td>REAR</td> <td>15</td> <td>16384</td> </tr> <tr> <td>SMPL</td> <td>8</td> <td>128</td> <td>REL</td> <td>16</td> <td>32768</td> </tr> </tbody> </table> <p>The following definitions exist: display.ANNUNCIATOR_x Where: x equals EDIT, ERROR, REMOTE, TALK, LISTEN, SRQ, REAR, REL, FILTER, MATH, 4_WIRE, AUTO, ARM, TRIGGER, STAR, or SAMPLE The values correspond to the annunciators listed above. For example: <pre>print(display.ANNUNCIATOR_EDIT) 2.560000000e+002 print(display.ANNUNCIATOR_TRIGGER) 3.200000000e+001 print(display.ANNUNCIATOR_AUTO) 8.000000000e+000</pre> </p>					Annunciator	Bit	Weighted Value	Annunciator	Bit	Weighted Value	FILT	1	1	EDIT	9	256	MATH	2	2	ERR	10	512	4W	3	4	REM	11	1024	AUTO	4	8	TALK	12	2048	ARM	5	16	LSTN	13	4096	TRIG	6	32	SRQ	14	8192	*(star)	7	64	REAR	15	16384	SMPL	8	128	REL	16	32768
Annunciator	Bit	Weighted Value	Annunciator	Bit	Weighted Value																																																						
FILT	1	1	EDIT	9	256																																																						
MATH	2	2	ERR	10	512																																																						
4W	3	4	REM	11	1024																																																						
AUTO	4	8	TALK	12	2048																																																						
ARM	5	16	LSTN	13	4096																																																						
TRIG	6	32	SRQ	14	8192																																																						
*(star)	7	64	REAR	15	16384																																																						
SMPL	8	128	REL	16	32768																																																						
Example	<p>Reads the annunciators that are turned on:</p> <pre>annun = display.getannunciators() print(annun)</pre> <p>Output: 1.280000e+03</p> <p>For the returned value of 1280, the binary equivalent is 0000010100000000. Bits 9 and 11 are set. Using the above chart in "Remarks", the REM and EDIT annunciators are turned on.</p>																																																										

display.getcursor()	
Function	Reads the present position of the cursor for the user display.
Usage	<pre>row, column, style = display.getcursor()</pre> <p>row: Returns the row for the present cursor position.</p> <p>column: Returns the column for the present cursor position.</p> <p>style: Returns the cursor style.</p>
Remarks	<ul style="list-style-type: none"> • 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).
Also see	<p>display.gettext() (on page 13-97)</p> <p>display.screen (on page 13-104)</p> <p>display.setcursor() (on page 13-105)</p> <p>display.settext() (on page 13-106)</p>
Example	<p>Reads cursor position (row and column):</p> <pre>row, column = display.getcursor() print(row, column)</pre> <p>Output: 1.000000e+00 3.000000e+00</p> <p>The above output indicates that the cursor is in Row 1 at Column 3.</p>
display.getlastkey()	
Function	Retrieves the key code for the last pressed key.
Usage	<pre>key = display.getlastkey()</pre>

display.getlastkey()																																																																									
Remarks	<p>This read-only function returns the key code for the last pressed key. <code>key</code> returns one of the following values:</p> <table border="1"> <thead> <tr> <th>Key List</th> <th>Value</th> <th>Key List</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>display.KEY_RIGHT</td> <td>103</td> <td>display.KEY_INSERT</td> <td>78</td> </tr> <tr> <td>display.KEY_LEFT</td> <td>104</td> <td>display.KEY_OPENALL</td> <td>79</td> </tr> <tr> <td>display.WHEEL_LEFT</td> <td>107</td> <td>display.KEY_CONFIG</td> <td>80</td> </tr> <tr> <td>display.WHEEL_RIGHT</td> <td>114</td> <td>display.KEY_RANGEDOWN</td> <td>81</td> </tr> <tr> <td>display.KEY_RANGEUP</td> <td>65</td> <td>display.KEY_ENTER</td> <td>82</td> </tr> <tr> <td>display.KEY_FUNC</td> <td>66</td> <td>display.KEY_REC</td> <td>83</td> </tr> <tr> <td>display.KEY_REL</td> <td>67</td> <td>display.KEY_DMM</td> <td>84</td> </tr> <tr> <td>display.KEY_MENU</td> <td>68</td> <td>display.KEY_DELETE</td> <td>85</td> </tr> <tr> <td>display.KEY_CLOSE</td> <td>69</td> <td>display.KEY_STEP</td> <td>86</td> </tr> <tr> <td>display.KEY_SLOT</td> <td>70</td> <td>display.KEY_CHAN</td> <td>87</td> </tr> <tr> <td>display.KEY_RUN</td> <td>71</td> <td>display.KEY_RATE</td> <td>90</td> </tr> <tr> <td>display.KEY_DISPLAY</td> <td>72</td> <td>display.KEY_LIMIT</td> <td>91</td> </tr> <tr> <td>display.KEY_AUTO</td> <td>73</td> <td>display.KEY_TRIG</td> <td>92</td> </tr> <tr> <td>display.KEY_FILTER</td> <td>74</td> <td>display.KEY_OPEN</td> <td>93</td> </tr> <tr> <td>display.KEY_EXIT</td> <td>75</td> <td>display.KEY_PATT</td> <td>94</td> </tr> <tr> <td>display.KEY_STORE</td> <td>76</td> <td>display.KEY_LOAD</td> <td>95</td> </tr> <tr> <td>display.KEY_SCAN</td> <td>77</td> <td>display.WHEEL_ENTER</td> <td>97</td> </tr> </tbody> </table> <p>A history of the key code for the last pressed front panel key is maintained by the Series 3700. When the instrument is powered-on, (or when transitioning from local to remote), the key code is set to 0 (display.KEY_NONE).</p> <p>Pressing the EXIT/LOCAL key normally aborts a script. In order to use this function with the EXIT key, display.locallockout (on page 13-102) must be used.</p>	Key List	Value	Key List	Value	display.KEY_RIGHT	103	display.KEY_INSERT	78	display.KEY_LEFT	104	display.KEY_OPENALL	79	display.WHEEL_LEFT	107	display.KEY_CONFIG	80	display.WHEEL_RIGHT	114	display.KEY_RANGEDOWN	81	display.KEY_RANGEUP	65	display.KEY_ENTER	82	display.KEY_FUNC	66	display.KEY_REC	83	display.KEY_REL	67	display.KEY_DMM	84	display.KEY_MENU	68	display.KEY_DELETE	85	display.KEY_CLOSE	69	display.KEY_STEP	86	display.KEY_SLOT	70	display.KEY_CHAN	87	display.KEY_RUN	71	display.KEY_RATE	90	display.KEY_DISPLAY	72	display.KEY_LIMIT	91	display.KEY_AUTO	73	display.KEY_TRIG	92	display.KEY_FILTER	74	display.KEY_OPEN	93	display.KEY_EXIT	75	display.KEY_PATT	94	display.KEY_STORE	76	display.KEY_LOAD	95	display.KEY_SCAN	77	display.WHEEL_ENTER	97
Key List	Value	Key List	Value																																																																						
display.KEY_RIGHT	103	display.KEY_INSERT	78																																																																						
display.KEY_LEFT	104	display.KEY_OPENALL	79																																																																						
display.WHEEL_LEFT	107	display.KEY_CONFIG	80																																																																						
display.WHEEL_RIGHT	114	display.KEY_RANGEDOWN	81																																																																						
display.KEY_RANGEUP	65	display.KEY_ENTER	82																																																																						
display.KEY_FUNC	66	display.KEY_REC	83																																																																						
display.KEY_REL	67	display.KEY_DMM	84																																																																						
display.KEY_MENU	68	display.KEY_DELETE	85																																																																						
display.KEY_CLOSE	69	display.KEY_STEP	86																																																																						
display.KEY_SLOT	70	display.KEY_CHAN	87																																																																						
display.KEY_RUN	71	display.KEY_RATE	90																																																																						
display.KEY_DISPLAY	72	display.KEY_LIMIT	91																																																																						
display.KEY_AUTO	73	display.KEY_TRIG	92																																																																						
display.KEY_FILTER	74	display.KEY_OPEN	93																																																																						
display.KEY_EXIT	75	display.KEY_PATT	94																																																																						
display.KEY_STORE	76	display.KEY_LOAD	95																																																																						
display.KEY_SCAN	77	display.WHEEL_ENTER	97																																																																						
Also see	<p>display.sendkey() (on page 13-105)</p> <p>display.locallockout (on page 13-102)</p>																																																																								
Example	<p>On the front panel, press the MENU key and then send the following code:</p> <pre>key = display.getlastkey() print(key)</pre> <p>Output: 6.800000e+01</p>																																																																								

display.gettext()	
Function	Reads the text presently displayed
Usage	<p>There are five ways to use this function:</p> <pre>text = display.gettext() text = display.gettext(embellished) text = display.gettext(embellished, row) text = display.gettext(embellished, row, column_start) text = display.gettext(embellished, row, column_start, column_end)</pre> <p>embellished Set to false to return text as a simple character string. Set to true to include all character codes.</p> <p>row Set to 1 or 2 to select which row to read text. If not included, text from both rows are read.</p> <p>column_start Set to starting column for reading text. Default is 1.</p> <p>column_end Set to ending column for reading text. Default is 20 (Row 1) or 32 (Row 2).</p> <hr/> <p>NOTE The range of valid column numbers depends on which row is specified. For Row 1, valid column numbers are 1 to 20. For Row 2, valid column numbers are 1 to 32.</p>
Remarks	<p>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.</p> <p>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 display.settext() (on page 13-106) for details on the character codes.</p> <p>The display will not be switched to the user screen. Text will be read from the active screen.</p>
Also see	<p>display.getcursor() (on page 13-95)</p> <p>display.setcursor() (on page 13-105)</p> <p>display.settext() (on page 13-106)</p>
Example	<p>Returns all text in both lines of the display:</p> <pre>text = display.gettext() print(text)</pre> <p>Output: User Screen \$N</p> <p>The above output indicates that the message "User Screen" is on the top line. The bottom line is blank.</p>

display.inputvalue()	
Function	Displays a formatted input field that the operator can edit.
Usage	<p>There are four ways to use this function:</p> <pre>value = display.inputvalue(format) value = display.inputvalue(format, default) value = display.inputvalue(format, default, min) value = display.inputvalue(format, default, min, max)</pre> <p>format: Define format string for the input field using `0's, the decimal point (.), polarity sign (+) and 'E' for exponent.</p> <p>default: Set the default value for the parameter.</p> <p>min: Set the minimum input value that can be set.</p> <p>max: Set the maximum input value that can be set.</p>
Remarks	<p>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 power-on will clear the user screen.</p> <p>Examples of the input field:</p> <ul style="list-style-type: none"> • +0.00 00 +00.0000E+00 • 0.00000E+0 <p>Value field:</p> <ul style="list-style-type: none"> • + Include a plus sign for positive/negative value entry. Do not include the "+" sign to prevents negative value entry. • 0 Defines the digit positions for the value. Up to six 0's can be used for the value (as shown above in the third and fourth examples). • . If used, include the decimal point (.) where needed for the value. <p>Exponent field (optional):</p> <ul style="list-style-type: none"> • E Include the "E" for exponent entry. • + Include a plus sign for positive/negative exponent entry. If a "+" sign is not included, negative exponent entry is prohibited. • 0 Defines the digit positions for the exponent. • You can also 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 (for example, -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 $\pm 1e37$.

display.inputvalue()	
Remarks (cont.)	<ul style="list-style-type: none"> • After sending this command, script execution pauses for the operator to enter a value and press the ENTER key. • 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 navigation wheel. Polarity will also toggle when using the navigation wheel to decrease or increase the value or exponent past zero. A zero value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity. • After sending this command and pressing the EXIT key, value will return nil.
Also see	display.prompt() (on page 13-102) display.setcursor() (on page 13-105) display.settext() (on page 13-106)
Example	Displays an editable field (" +0.50") for operator input. Valid input range: 0.10 to +2.00, with a default of 0.50: <pre>display.clear() value = display.inputvalue("+0.00", 0.5, -0.1, 2.0)</pre>

display.loadmenu.add()	
Function	Adds an entry to the USER TESTS submenu of the LOAD TEST menu.
Usage	There are two ways to use this function: <pre>display.loadmenu.add(displayname, chunk) display.loadmenu.add(displayname, chunk, memory)</pre> <p>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 <code>display.DONT_SAVE</code> 1 or <code>display.SAVE</code> The default memory setting is <code>display.SAVE</code></p>

display.loadmenu.add()	
Remarks	<ul style="list-style-type: none"> This function adds an entry to the USER TESTS submenu of the LOAD TEST menu. If the given item is subsequently selected using 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 <code>display.SAVE</code>, commands are saved with the chunk in non-volatile memory. Scripts, functions, and variables used in the chunk are not saved by <code>display.SAVE</code>. Functions and variables need to be saved along with the script. If the script is not saved in non-volatile memory, it will be lost when the Series 3700 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.
Also see	display.loadmenu.delete() (on page 13-101)
Examples	<p>Example 1: Assume a script with a function named 'DUT1' has already been loaded into the Series 3700, and the script has NOT been saved in non-volatile memory.</p> <p>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:</p> <pre>display.loadmenu.add('Test', 'DUT1() beeper.beep(2, 500)', display.SAVE)</pre> <p>When 'Test' is run from the front panel USER TESTS menu, the function named 'DUT1' will execute and the beeper will beep for two seconds.</p> <p>Now assume you cycle power on the Series 3700. Because the script was not saved in non-volatile memory, the function named 'DUT1' is lost. When 'Test' is again run from the front panel, the beeper will beep, but 'DUT1' will not execute because it no longer exists in the chunk.</p> <p>Example 2: This example adds an entry called 'Part1' to the front panel USER TESTS load menu for the chunk 'testpart([[Part1]], 5.0)', and saves it in non-volatile memory:</p> <pre>display.loadmenu.add('Part1', 'testpart([[Part1]], 5.0)', display.SAVE)</pre>

display.loadmenu.delete()	
Function	Deletes an entry from the USER submenu of the LOAD TEST menu.
Usage	<code>display.loadmenu.delete(displayname)</code> displayname: Name to remove from the menu.
Remarks	This function is used to delete an entry (<code>displayname</code>) from the front panel USER TESTS submenu of the LOAD TEST menu.
Also see	display.loadmenu.add() (on page 13-100)
Example	Removes the entry named 'Part1' from the front panel USER TESTS load menu: <code>display.loadmenu.delete('Part1')</code>

display.locallockout	
Attribute	LOCAL key disabled.
Usage	To read state of lockout: <code>lockout = display.locallockout</code> To write state of lockout: <code>display.locallockout = lockout</code> Set lockout to one of the following values: Unlocks LOCAL key: 0 or <code>display.UNLOCK</code> Locks out LOCAL key: 1 or <code>display.LOCK</code>
Remarks	Setting <code>display.locallockout</code> to <code>display.LOCK</code> prevents the user from interrupting remote operation by pressing the LOCAL key. Set this attribute to <code>display.UNLOCK</code> to allow the LOCAL key to abort script/remote operation.
Example	Disables the front panel LOCAL key: <code>display.locallockout = display.LOCK</code>
display.menu()	
Function	Presents a menu on the front panel display.
Usage	<code>selection = display.menu(name, items)</code> name : Menu name to display on the top line. items : Menu items to display on the bottom line.
Remarks	<ul style="list-style-type: none"> The menu consists of the menu name string on the top line, and a selectable list of items on the bottom line. The menu items must be a single string with each item separated by whitespace. The name for the top line is limited to 20 characters. After sending this command, script execution pauses for the operator to select a menu item. An item is selected by rotating the navigation wheel (or using the CURSOR keys) to place the blinking cursor on the item, and then pressing the navigation 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.
Example	Displays a menu with three menu items. If the second menu item is selected, selection will be given the value <code>Test2</code> : <code>selection = display.menu("Sample Menu", "Test1 Test2 Test3")</code> <code>print(selection)</code> Output: <code>Test2</code>

display.prompt()	
Function	Prompts the user to enter a parameter from the front panel.
Usage	<p>There are four ways to use this function:</p> <pre>value = display.prompt(format, units, help) value = display.prompt(format, units, help, default) value = display.prompt(format, units, help, default, min) value = display.prompt(format, units, help, default, min, max)</pre> <p>format: Define format string for the input field using zeros (0), the decimal point (.), polarity sign (+), and exponent (E).</p> <p>units: Set units text string for top line (8 characters maximum).</p> <p>help: Text string to display on the bottom line (32 characters maximum).</p> <p>default: Set the default value for the parameter.</p> <p>min: Set the minimum input value that can be set.</p> <p>max: Set the maximum input value that can be set.</p>
Remarks	<p>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:</p> <ul style="list-style-type: none"> • 0.00V • Input 0 to +2V <p>The format configures the editable input field. Four examples for the format:</p> <ul style="list-style-type: none"> • +0.00 • 00 • +00.0000E+00 • 0.00000E+0 <p>Value field:</p> <ul style="list-style-type: none"> • +: Include a plus 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's can be used for the value (as shown above in the third and fourth usage examples). If used, include the decimal point (.) where needed for the value. <p>Exponent field (optional):</p> <ul style="list-style-type: none"> • E: Include the "E" for exponent entry. • +: Include a "+" sign for positive/negative exponent entry. Do not include the "+" sign to prevents negative value entry. • 0: Defines the digit positions for the exponent.

display.prompt()	
Remarks (cont.)	<ul style="list-style-type: none"> • <code>units</code>: a string that indicates the units (for example, "V" or "A") for the value and help provides a message prompt on the bottom line. • You can also 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 (for example, -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 $\pm 1e37$. • After sending this command, script execution pauses 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 navigation wheel. Polarity will also toggle when using the navigation wheel to decrease or increase the value or exponent past zero. A zero value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity. • After sending this command and pressing the EXIT key, the value will return nil.
Also see	<i>display.inputvalue()</i> (on page 13-98)
Example	<p>Prompts the operator to enter a voltage value; valid input range is 0 to +2.00, with a default of 0.50:</p> <pre>value = display.prompt("0.00", "v", "Input 0 to +2V" 0.5, 0, 2)</pre> <p>The above command will display the following input prompt:</p> <pre>0.50V Input 0 to +2V</pre>

display.screen	
Attribute	The selected display screen.
Usage	<p>To read display screen: <code>displayid = display.screen</code></p> <p>To write display screen: <code>display.screen = displayid</code></p> <p>Set displayid to one of the following values:</p> <p>0 or <code>display.MAIN</code>: Displays main screen.</p> <p>1 or <code>display.USER</code>: Displays the user screen.</p>
Remarks	Setting this attribute selects the display screen for the front panel. Read this attribute to determine which of the available display screens was last selected.

display.screen																																																																									
Example	Selects the user display: <code>display.screen = display.USER</code>																																																																								
display.sendkey()																																																																									
Function	Send key code that simulates the action of a front panel control being pressed.																																																																								
Usage	<p><code>display.sendkey(keycode)</code></p> <p>Set key code to one of the values shown below:</p> <table border="1"> <thead> <tr> <th>Key List</th> <th>Value</th> <th>Key List</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>display.KEY_RIGHT</code></td> <td>103</td> <td><code>display.KEY_INSERT</code></td> <td>78</td> </tr> <tr> <td><code>display.KEY_LEFT</code></td> <td>104</td> <td><code>display.KEY_OPENALL</code></td> <td>79</td> </tr> <tr> <td><code>display.WHEEL_LEFT</code></td> <td>107</td> <td><code>display.KEY_CONFIG</code></td> <td>80</td> </tr> <tr> <td><code>display.WHEEL_RIGHT</code></td> <td>114</td> <td><code>display.KEY_RANGEDO WN</code></td> <td>81</td> </tr> <tr> <td><code>display.KEY_RANGEUP</code></td> <td>65</td> <td><code>display.KEY_ENTER</code></td> <td>82</td> </tr> <tr> <td><code>display.KEY_FUNC</code></td> <td>66</td> <td><code>display.KEY_REC</code></td> <td>83</td> </tr> <tr> <td><code>display.KEY_REL</code></td> <td>67</td> <td><code>display.KEY_DMM</code></td> <td>84</td> </tr> <tr> <td><code>display.KEY_MENU</code></td> <td>68</td> <td><code>display.KEY_DELETE</code></td> <td>85</td> </tr> <tr> <td><code>display.KEY_CLOSE</code></td> <td>69</td> <td><code>display.KEY_STEP</code></td> <td>86</td> </tr> <tr> <td><code>display.KEY_SLOT</code></td> <td>70</td> <td><code>display.KEY_CHAN</code></td> <td>87</td> </tr> <tr> <td><code>display.KEY_RUN</code></td> <td>71</td> <td><code>display.KEY_RATE</code></td> <td>90</td> </tr> <tr> <td><code>display.KEY_DISPLAY</code></td> <td>72</td> <td><code>display.KEY_LIMIT</code></td> <td>91</td> </tr> <tr> <td><code>display.KEY_AUTO</code></td> <td>73</td> <td><code>display.KEY_TRIG</code></td> <td>92</td> </tr> <tr> <td><code>display.KEY_FILTER</code></td> <td>74</td> <td><code>display.KEY_OPEN</code></td> <td>93</td> </tr> <tr> <td><code>display.KEY_EXIT</code></td> <td>75</td> <td><code>display.KEY_PATT</code></td> <td>94</td> </tr> <tr> <td><code>display.KEY_STORE</code></td> <td>76</td> <td><code>display.KEY_LOAD</code></td> <td>95</td> </tr> <tr> <td><code>display.KEY_SCAN</code></td> <td>77</td> <td><code>display.WHEEL_ENTER</code></td> <td>97</td> </tr> </tbody> </table> <p>NOTE When using this function, send built-in defines such as <code>display.KEY_RIGHT</code> (rather than the numeric value of 103). This will allow for greater forward compatibility with respect to firmware revisions. For example, use <code>display.KEY_ENTER</code> instead of 82.</p>	Key List	Value	Key List	Value	<code>display.KEY_RIGHT</code>	103	<code>display.KEY_INSERT</code>	78	<code>display.KEY_LEFT</code>	104	<code>display.KEY_OPENALL</code>	79	<code>display.WHEEL_LEFT</code>	107	<code>display.KEY_CONFIG</code>	80	<code>display.WHEEL_RIGHT</code>	114	<code>display.KEY_RANGEDO WN</code>	81	<code>display.KEY_RANGEUP</code>	65	<code>display.KEY_ENTER</code>	82	<code>display.KEY_FUNC</code>	66	<code>display.KEY_REC</code>	83	<code>display.KEY_REL</code>	67	<code>display.KEY_DMM</code>	84	<code>display.KEY_MENU</code>	68	<code>display.KEY_DELETE</code>	85	<code>display.KEY_CLOSE</code>	69	<code>display.KEY_STEP</code>	86	<code>display.KEY_SLOT</code>	70	<code>display.KEY_CHAN</code>	87	<code>display.KEY_RUN</code>	71	<code>display.KEY_RATE</code>	90	<code>display.KEY_DISPLAY</code>	72	<code>display.KEY_LIMIT</code>	91	<code>display.KEY_AUTO</code>	73	<code>display.KEY_TRIG</code>	92	<code>display.KEY_FILTER</code>	74	<code>display.KEY_OPEN</code>	93	<code>display.KEY_EXIT</code>	75	<code>display.KEY_PATT</code>	94	<code>display.KEY_STORE</code>	76	<code>display.KEY_LOAD</code>	95	<code>display.KEY_SCAN</code>	77	<code>display.WHEEL_ENTER</code>	97
Key List	Value	Key List	Value																																																																						
<code>display.KEY_RIGHT</code>	103	<code>display.KEY_INSERT</code>	78																																																																						
<code>display.KEY_LEFT</code>	104	<code>display.KEY_OPENALL</code>	79																																																																						
<code>display.WHEEL_LEFT</code>	107	<code>display.KEY_CONFIG</code>	80																																																																						
<code>display.WHEEL_RIGHT</code>	114	<code>display.KEY_RANGEDO WN</code>	81																																																																						
<code>display.KEY_RANGEUP</code>	65	<code>display.KEY_ENTER</code>	82																																																																						
<code>display.KEY_FUNC</code>	66	<code>display.KEY_REC</code>	83																																																																						
<code>display.KEY_REL</code>	67	<code>display.KEY_DMM</code>	84																																																																						
<code>display.KEY_MENU</code>	68	<code>display.KEY_DELETE</code>	85																																																																						
<code>display.KEY_CLOSE</code>	69	<code>display.KEY_STEP</code>	86																																																																						
<code>display.KEY_SLOT</code>	70	<code>display.KEY_CHAN</code>	87																																																																						
<code>display.KEY_RUN</code>	71	<code>display.KEY_RATE</code>	90																																																																						
<code>display.KEY_DISPLAY</code>	72	<code>display.KEY_LIMIT</code>	91																																																																						
<code>display.KEY_AUTO</code>	73	<code>display.KEY_TRIG</code>	92																																																																						
<code>display.KEY_FILTER</code>	74	<code>display.KEY_OPEN</code>	93																																																																						
<code>display.KEY_EXIT</code>	75	<code>display.KEY_PATT</code>	94																																																																						
<code>display.KEY_STORE</code>	76	<code>display.KEY_LOAD</code>	95																																																																						
<code>display.KEY_SCAN</code>	77	<code>display.WHEEL_ENTER</code>	97																																																																						
Also see	Front panel keys for key functions.																																																																								
Example	To use a bus command to simulate the action of a front panel RUN key being pressed: <code>display.sendkey(display.KEY_RUN)</code>																																																																								

display.setcursor()	
Function	Sets the position of the cursor.
Usage	<p>There are two ways to use this function:</p> <pre>display.setcursor(row, column) display.setcursor(row, column, style)</pre> <p>row: Set row number for the cursor (1 or 2).</p> <p>column: Set the column number for the cursor. For row 1, the column can be set from 1 to 20. For row 2, the column can be set from 1 to 32.</p> <p>style: Set the cursor style to be invisible (0) or blink (1).</p>
Remarks	<ul style="list-style-type: none"> • 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 <i>style</i> (a parameter other than 1 or 0) 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() (on page 13-93), <code>display.setcursor</code>, and display.settext() (on page 13-106) 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.
Also see	<p>display.clear() (on page 13-93)</p> <p>display.getcursor() (on page 13-95)</p> <p>display.gettext() (on page 13-97)</p> <p>display.settext() (on page 13-106)</p>
Example	<p>Positions cursor in row 2, column 1:</p> <pre>display.setcursor(2, 1)</pre>
display.settext()	
Function	Displays text on the user screen.
Usage	<pre>display.settext("text")</pre> <p>text: Text message string to be displayed.</p>

display.settext()	
Remarks	<p>This function selects the user display screen, and displays the given text. The first write to the display after power-on will clear the user screen.</p> <p>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.</p> <p>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.</p> <p>The text remains on the display until replaced or cleared.</p> <p>The following character codes can be also be included in the text string:</p> <ul style="list-style-type: none"> • \$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: Sets text to background blink. • \$\$: Escape sequence to display a single "\$". <p>The display.clear (on page 13-93), display.setcursor (on page 13-105), 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.</p>
Also see	<p>display.clear() (on page 13-93)</p> <p>display.getcursor() (on page 13-95)</p> <p>display.gettext() (on page 13-97)</p> <p>display.setcursor() (on page 13-105)</p>
Example	<p>Displays a message on the user screen:</p> <p>To display "Message Test" on the top line and the bottom line displays the blinking message "with Row 2 Blinking":</p> <pre>display.clear() display.settext("Message Test \$N\$Bwith Row 2 Blinking")</pre> <p>To display the unit's serial number on the top line:</p> <pre>display.clear() display.settext(localnode.serialno)</pre> <p>(see localnode.serialno (on page 13-214))</p>

display.waitkey()	
Function	Captures the key code value for the next key press.
Usage	<code>key = display.waitkey()</code>

display.waitkey()																																																																									
Remarks	<p>After sending this function, script execution pauses until a front panel key or the navigation wheel is pressed, or the navigation wheel is turned to the right or left. After pressing a control or turning the navigation wheel, the key code value for that key will be returned. The chart shown below lists the key code value for each front panel control. The controls are listed alphabetically.</p> <p>If the EXIT key is pressed while this function is waiting for a key press, the script will not be aborted.</p> <p>A typical use for this function is to prompt the user to press the EXIT key to abort the script or press any other key to continue. If key code value 75 is returned (EXIT key pressed), then the <code>exit()</code> function can be called to abort the script.</p>																																																																								
	<table border="1"> <thead> <tr> <th>Key List</th> <th>Value</th> <th>Key List</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td><code>display.KEY_RIGHT</code></td> <td>103</td> <td><code>display.KEY_INSERT</code></td> <td>78</td> </tr> <tr> <td><code>display.KEY_LEFT</code></td> <td>104</td> <td><code>display.KEY_OPENALL</code></td> <td>79</td> </tr> <tr> <td><code>display.WHEEL_LEFT</code></td> <td>107</td> <td><code>display.KEY_CONFIG</code></td> <td>80</td> </tr> <tr> <td><code>display.WHEEL_RIGHT</code></td> <td>114</td> <td><code>display.KEY_RANGEDO WN</code></td> <td>81</td> </tr> <tr> <td><code>display.KEY_RANGEUP</code></td> <td>65</td> <td><code>display.KEY_ENTER</code></td> <td>82</td> </tr> <tr> <td><code>display.KEY_FUNC</code></td> <td>66</td> <td><code>display.KEY_REC</code></td> <td>83</td> </tr> <tr> <td><code>display.KEY_REL</code></td> <td>67</td> <td><code>display.KEY_DMM</code></td> <td>84</td> </tr> <tr> <td><code>display.KEY_MENU</code></td> <td>68</td> <td><code>display.KEY_DELETE</code></td> <td>85</td> </tr> <tr> <td><code>display.KEY_CLOSE</code></td> <td>69</td> <td><code>display.KEY_STEP</code></td> <td>86</td> </tr> <tr> <td><code>display.KEY_SLOT</code></td> <td>70</td> <td><code>display.KEY_CHAN</code></td> <td>87</td> </tr> <tr> <td><code>display.KEY_RUN</code></td> <td>71</td> <td><code>display.KEY_RATE</code></td> <td>90</td> </tr> <tr> <td><code>display.KEY_DISPLAY</code></td> <td>72</td> <td><code>display.KEY_LIMIT</code></td> <td>91</td> </tr> <tr> <td><code>display.KEY_AUTO</code></td> <td>73</td> <td><code>display.KEY_TRIG</code></td> <td>92</td> </tr> <tr> <td><code>display.KEY_FILTER</code></td> <td>74</td> <td><code>display.KEY_OPEN</code></td> <td>93</td> </tr> <tr> <td><code>display.KEY_EXIT</code></td> <td>75</td> <td><code>display.KEY_PATT</code></td> <td>94</td> </tr> <tr> <td><code>display.KEY_STORE</code></td> <td>76</td> <td><code>display.KEY_LOAD</code></td> <td>95</td> </tr> <tr> <td><code>display.KEY_SCAN</code></td> <td>77</td> <td><code>display.WHEEL_ENTER</code></td> <td>97</td> </tr> </tbody> </table> <p>The above chart lists the numeric key code values for the front panel controls. The key code value identifiers are listed in the documentation for display.sendkey() (on page 13-105) (for example, <code>display.KEY_RUN</code> is the identifier for the RUN key).</p>	Key List	Value	Key List	Value	<code>display.KEY_RIGHT</code>	103	<code>display.KEY_INSERT</code>	78	<code>display.KEY_LEFT</code>	104	<code>display.KEY_OPENALL</code>	79	<code>display.WHEEL_LEFT</code>	107	<code>display.KEY_CONFIG</code>	80	<code>display.WHEEL_RIGHT</code>	114	<code>display.KEY_RANGEDO WN</code>	81	<code>display.KEY_RANGEUP</code>	65	<code>display.KEY_ENTER</code>	82	<code>display.KEY_FUNC</code>	66	<code>display.KEY_REC</code>	83	<code>display.KEY_REL</code>	67	<code>display.KEY_DMM</code>	84	<code>display.KEY_MENU</code>	68	<code>display.KEY_DELETE</code>	85	<code>display.KEY_CLOSE</code>	69	<code>display.KEY_STEP</code>	86	<code>display.KEY_SLOT</code>	70	<code>display.KEY_CHAN</code>	87	<code>display.KEY_RUN</code>	71	<code>display.KEY_RATE</code>	90	<code>display.KEY_DISPLAY</code>	72	<code>display.KEY_LIMIT</code>	91	<code>display.KEY_AUTO</code>	73	<code>display.KEY_TRIG</code>	92	<code>display.KEY_FILTER</code>	74	<code>display.KEY_OPEN</code>	93	<code>display.KEY_EXIT</code>	75	<code>display.KEY_PATT</code>	94	<code>display.KEY_STORE</code>	76	<code>display.KEY_LOAD</code>	95	<code>display.KEY_SCAN</code>	77	<code>display.WHEEL_ENTER</code>	97
Key List	Value	Key List	Value																																																																						
<code>display.KEY_RIGHT</code>	103	<code>display.KEY_INSERT</code>	78																																																																						
<code>display.KEY_LEFT</code>	104	<code>display.KEY_OPENALL</code>	79																																																																						
<code>display.WHEEL_LEFT</code>	107	<code>display.KEY_CONFIG</code>	80																																																																						
<code>display.WHEEL_RIGHT</code>	114	<code>display.KEY_RANGEDO WN</code>	81																																																																						
<code>display.KEY_RANGEUP</code>	65	<code>display.KEY_ENTER</code>	82																																																																						
<code>display.KEY_FUNC</code>	66	<code>display.KEY_REC</code>	83																																																																						
<code>display.KEY_REL</code>	67	<code>display.KEY_DMM</code>	84																																																																						
<code>display.KEY_MENU</code>	68	<code>display.KEY_DELETE</code>	85																																																																						
<code>display.KEY_CLOSE</code>	69	<code>display.KEY_STEP</code>	86																																																																						
<code>display.KEY_SLOT</code>	70	<code>display.KEY_CHAN</code>	87																																																																						
<code>display.KEY_RUN</code>	71	<code>display.KEY_RATE</code>	90																																																																						
<code>display.KEY_DISPLAY</code>	72	<code>display.KEY_LIMIT</code>	91																																																																						
<code>display.KEY_AUTO</code>	73	<code>display.KEY_TRIG</code>	92																																																																						
<code>display.KEY_FILTER</code>	74	<code>display.KEY_OPEN</code>	93																																																																						
<code>display.KEY_EXIT</code>	75	<code>display.KEY_PATT</code>	94																																																																						
<code>display.KEY_STORE</code>	76	<code>display.KEY_LOAD</code>	95																																																																						
<code>display.KEY_SCAN</code>	77	<code>display.WHEEL_ENTER</code>	97																																																																						
Also see	<p>display.sendkey() (on page 13-105)</p> <p>display.settext() (on page 13-106)</p> <p>display.getlastkey() (on page 13-96)</p>																																																																								

display.waitkey()	
Example	<p>The following code pauses script execution and wait for the operator to press a key or the navigation wheel, or rotate the navigation wheel:</p> <pre>key = display.waitkey() print(key) → 8.600000e+01</pre> <p>The above output (8.600000e+01, or 86) indicates that the STEP key was pressed.</p>

dmm functions and attributes

Use the functions and attributes in this group to control DMM operation, set limits, and perform calibration. Default configuration names like "dcvolts", "temperature", "fourwireohms", etc., imply the reset conditions for the corresponding functions. For example, sending `dmm.setconfig("slot1", "dcvolts")` sets all channels on Slot 1 to use DC volts factory-default settings. To make a channel use one setting that is different, create a user dmm configuration with `dmm.configure.set("mydcv")` and then set "mydcv" to the desired channels.

dmm.adjustment.count	
Attribute	Indicates the number of times the unit has been adjusted.
Usage	<pre>cal_count = dmm.adjustment.count</pre> <p>cal_count: Represents the number of times unit has been adjusted.</p>
Remarks	<p>Use this attribute to query the unit for the number of times the unit has been adjusted.</p> <p>This item can only be set when calibration is unlocked.</p>
Example	<p>To query for the adjustment count:</p> <pre>CalCount = dmm.adjustment.count</pre>

dmm.adjustment.date	
Attribute	Sets or queries the adjustment date in UTC format (number of seconds since January 1, 1970).
Usage	<pre>CalDate = dmm.adjustment.date</pre> <p>CalDate: Represents the number of seconds since January 1, 1970.</p> <p>To set the adjustment date based on the present date of the system:</p> <pre>dmm.adjustment.date = os.time()</pre> <p>To set the adjustment date as July 4, 2007</p> <pre>dmm.adjustment.date = os.time({year=2007, month=7, day = 4})</pre>
Remarks	<p>Use this attribute to set and get the adjustment date of the DMM in UTC format. See Lua documentation for formatting options with <code>os.date</code>.</p> <p>This item can only be set when calibration is unlocked.</p>

dmm.adjustment.date	
Example	<p>Also see usage for setting date.</p> <hr/> <p>NOTE For following assume set date to July 4, 2007.</p> <hr/> <p>To query date and format the response as mm/dd/yyyy: <pre>print(os.date("%m/%d/%Y", dmm.adjustment.date)) → 07/04/2007</pre> </p> <p>To query date and format the response as mm/dd/yy: <pre>print(os.date("%x", dmm.adjustment.date)) → 07/04/07</pre> </p>

dmm.aperture									
Attribute	Indicates the aperture setting for the active DMM function in seconds.								
Usage	<p>To read the aperture: <pre>value = dmm.aperture</pre> value: Represents the present aperture setting in seconds</p> <p>To write the aperture: <pre>dmm.aperture = value</pre> value: Represents the desired aperture:</p> <ul style="list-style-type: none"> • For 50Hz, the range is 10e-6 to 0.250 second. • For 60Hz, the range is 8.33e-6 to 0.250 second. • For frequency and period, 0.01 to 0.273 second. 								
Remarks	<p>This is the aperture setting for the DMM and it applies to the selected function as indicated by <i>dmm.func</i> (on page 13-137). Querying the aperture when the selected function does not have an aperture associated with it will cause nil to be returned.</p> <p>An error is generated if a command is received when <i>dmm.func</i> (on page 13-137) = "continuity" or "nofunction".</p> <p>Changing functions with <i>dmm.func</i> (on page 13-137) will reflect the aperture setting for that function.</p> <p>The setting for aperture may be adjusted based on what the DMM supports. Therefore, after setting the aperture, query the value to see if it was adjusted.</p> <p>If the detector bandwidth (<i>dmm.detectorbandwidth</i> (on page 13-132)) setting is 30 or less for "acvolts" or "accurrent", then an error message will be generated when trying to set the aperture for these functions. The detector bandwidth setting is one of three values (based on input parameter): 3, 30, or 300.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 50%;">Input parameter</th> <th style="width: 50%;">Converted value</th> </tr> </thead> <tbody> <tr> <td>< 30</td> <td>3</td> </tr> <tr> <td>Between 30 and 300</td> <td>30</td> </tr> <tr> <td>≥ 300</td> <td>300</td> </tr> </tbody> </table>	Input parameter	Converted value	< 30	3	Between 30 and 300	30	≥ 300	300
Input parameter	Converted value								
< 30	3								
Between 30 and 300	30								
≥ 300	300								

dmm.aperture	
Also see	dmm.func (on page 13-137) dmm.nplc (on page 13-152)
Example	To set the aperture to 16.67 milliseconds for DC volts: <pre>dmm.func = "dcvolts" dmm.aperture = 16.67e-3</pre>

dmm.appendbuffer()	
Function	Appends data from reading buffer to USB flash drive.
Usage	<pre>dmm.appendbuffer('<reading buffer name>', '<filename>', time_format)</pre> <p>reading buffer name: The name of a previously created DMM reading buffer.</p> <p>filename: The destination filename located on the USB flash drive.</p> <p>time_format: This optional parameter indicates how the date and time information should be appended to the file to the thumb drive. Use the following values for time_format:</p> <ul style="list-style-type: none"> • <code>dmm.buffer.SAVE_RELATIVE_TIME</code>, which saves relative time stamps only • <code>dmm.buffer.SAVE_FORMAT_TIME</code>, which is the default if no time format specified and saves dates, times and fractional seconds • <code>dmm.buffer.SAVE_RAW_TIME</code>, which saves seconds and fractionalseconds only • <code>dmm.buffer.SAVE_TIMESTAMP_TIME</code>, which only saves time stamps

dmm.appendbuffer()	
Remarks	<p>The first parameter (<code>reading buffer name</code>) represents the reading buffer to be saved. The second (<code>filename</code>) is the filename of file to append reading buffer data to on USB flash drive. The third parameter is optional and indicates how the date and time information from the buffer should be saved. For options that save more than one item of time information, each item is comma delimited. For example, the default format will have <code><date></code>, <code><time></code>, and <code><fractional seconds></code> for each reading, separated by commas.</p> <p>This command appends data to an existing file or creates the file, if it doesn't exist. Because it is appending data, it does not include the header information like the dmm.savebuffer() (on page 7-10) command. This command doesn't care about the data format of existing data because it only appends data. Therefore, the index column entry starts at 1 for each append operation. Like the <code>savebuffer</code> command, it appends all data in the reading buffer specified.</p> <p>Errors are generated if the reading buffer does not exist or is not a DMM buffer, or if the destination filename is not specified correctly. The file extension <code>.csv</code> is appended to the filename if necessary. Any specified file extension other than <code>.csv</code> generates errors.</p> <p>Valid destination filename examples:</p> <pre>dmm.appendbuffer('mybuffer', '/usb1/mydata')</pre> <pre>dmm.appendbuffer('mybuffer', '/usb1/mydata.csv')</pre> <p>Invalid destination filename examples:</p> <pre>dmm.appendbuffer('mybuffer', '/usb1/mydata.')</pre> <p>-Invalid extension due to period by no following letters for extension.</p> <pre>dmm.appendbuffer('mybuffer', '/usb1/mydata.txt')</pre> <p>-Invalid extension. Use <code>.csv</code> or do not specify (no period)</p> <pre>dmm.appendbuffer('mybuffer', '/usb1/mydata.txt.csv')</pre> <p>-invalid extension because 2 periods specified (<code>mydata_txt.csv</code> would be correct).</p> <hr/> <p>NOTE The reading buffer files saved to the USB flash drive will always have an extension of <code>.csv</code></p>
Example	<p>To append readings from valid DMM buffer named <code>mybuffer</code> with default time information to a file named <code>mydata.csv</code> on the USB flash drive:</p> <pre>dmm.appendbuffer('mybuffer', '/usb1/mydata.csv')</pre> <p>To append readings from <code>mybuffer</code> with relative time stamps to a file named <code>mydatarel.csv</code> on the USB flash drive:</p> <pre>dmm.appendbuffer('mybuffer', '/usb1/mydatarel.csv', dmm.buffer.SAVE_RELATIVE_TIME)</pre>

dmm.autodelay	
Attribute	Indicates the auto delay setting for the active DMM function.
Usage	<p>To read the autodelay setting: <code>value = dmm.autodelay</code></p> <p>value: Represents the present auto delay setting</p> <p>To write the autodelay setting: <code>dmm.autodelay = value</code></p> <p>value: Represents the desired auto delay. Set to one of the following:</p> <ul style="list-style-type: none"> • <code>dmm.ON</code> or 1 to enable auto delay • <code>dmm.OFF</code> or 0 to disable auto delay • <code>dmm.AUTODELAY_ONCE</code> or 2
Remarks	<p>This is the auto delay setting for the DMM and it applies to the selected function as indicated by dmm.func (on page 13-137). Querying the auto delay when the selected function doesn't have an auto delay setting associated with it will cause nil to be returned.</p> <p>An error is generated if command is received when <code>dmm.func = "nofunction"</code>. Also, an error will be generated if the value is invalid.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the auto delay setting for that function.</p> <p>A setting of <code>dmm.OFF</code> has the DMM take a measurement as fast as possible without concern of delays needed based on range, function and other settings.</p> <p>A setting of <code>dmm.ON</code> has the DMM wait the necessary delay for each and every measurement it takes.</p> <p>A setting of <code>dmm.AUTODELAY_ONCE</code> will only incur the delay on the first measurement set or group. This is the factory default and reset setting.</p> <p>For example:</p> <pre>dmm.autodelay = dmm.AUTODELAY_ONCE dmm.measurecount = 10 MyReadingBuffer = dmm.makebuffer(1000) dmm.measure(MyReadingBuffer)</pre> <p>This will only incur the auto delay on the first of the 10 readings. Readings 2 through 10 will occur as fast as possible.</p>
Also see	dmm.func (on page 13-137)
Example	<p>To enable auto delay for DC volts:</p> <pre>dmm.func = "dcvolts" dmm.autodelay = dmm.ON</pre> <p>To set auto delay to set once:</p> <pre>dmm.autodelay = dmm.AUTODELAY_ONCE</pre>

dmm.autorange	
Attribute	Indicates the auto range setting for the active DMM function.
Usage	<p>To read the auto range:</p> <pre>value = dmm.autorange</pre> <p>value: Represents the present auto range setting (1 = ON, 0 = OFF).</p> <p>To write the auto range:</p> <pre>dmm.autorange = value</pre> <p>value: Represents the desired auto range setting. Use one of the following:</p> <ul style="list-style-type: none"> • dmm.ON or 1: Enables auto ranging. • dmm.OFF or 0: Disables auto ranging.
Remarks	<p>This is the auto range setting for the DMM. It applies to the selected function as indicated by <i>dmm.func</i> (on page 13-137). Querying the auto range when the selected function does not have an auto range associated with it will cause nil to be returned.</p> <p>An error is generated if command is received when <i>dmm.func</i> = "temperature", "frequency", "period", "continuity" or "nofunction". Also, an error will be generated if the value is out of range.</p> <p>Changing functions with <i>dmm.func</i> (on page 13-137) reflects the autorange setting for that function.</p> <p>The default setting is dmm.ON.</p>
Example	<p>To enable auto ranging for 2-wire ohms:</p> <pre>dmm.func = "twowireohms"</pre> <pre>dmm.autorange = dmm.ON</pre>

dmm.autozero	
Attribute	Indicates the auto zero setting for the active DMM function.
Usage	<p>To read the auto zero status:</p> <pre>value = dmm.autozero</pre> <p>value: Represents the present auto zero setting (1 = ON, 0 = OFF)</p> <p>To change the auto zero status:</p> <pre>dmm.autozero = value</pre> <p>value: Represents the desired auto zero. Use one of the following:</p> <ul style="list-style-type: none"> • dmm.ON or 1 to enable auto zero • dmm.OFF or 0 to disable auto zero • dmm.AUTOZERO_ONCE or 2 to refresh backgrounds once and go to auto zero off setting (0 will be returned if auto zero status is read).

dmm.autozero	
Remarks	<p>This is the auto zero setting for the DMM. It applies to the selected function as indicated by <i>dmm.func</i> (on page 13-137). Querying the auto zero when the selected function does not have an auto zero setting associated with it will cause nil to be returned.</p> <p>An error is generated if command is received when <code>dmm.func = "nofunction"</code>. Also, an error will be generated if the value is invalid.</p> <p>Changing functions with <i>dmm.func</i> (on page 13-137) will cause that function's auto zero setting to take effect.</p> <p>The auto zero once setting will force a refresh of the backgrounds. After doing this once, the auto zero setting will automatically be set to "OFF". Therefore, querying the <code>dmm.autozero</code> state after setting <code>dmm.autozero</code> to 2 generates a response of 0 and not 2.</p>
Example	<p>To enable auto zero for DC volts:</p> <pre>dmm.func = "dcvolts" dmm.autozero = dmm.ON</pre> <p>To force backgrounds to cycle once and set auto zero to OFF:</p> <pre>dmm.autozero = dmm.AUTOZERO_ONCE print(dmm.autozero) → 0.000000000e+000</pre>

dmm.buffer.catalog()	
Function	Creates an iterator for the user-created reading buffers.
Usage	<code>for name in dmm.buffer.catalog() do...end</code>
Remarks	<p>Accessing the catalog for the user-created local reading buffers allows the user to print the names of all reading buffers in system. The entries will be enumerated in no particular order. From this list, one may selectively delete reading buffers from the system. For example:</p> <pre>for name in dmm.buffer.catalog() do print(name) end</pre> <p>outputs:</p> <pre>buf3 buf5 buf1</pre> <p>Now, with these buffers in the system, to delete <code>buf1</code>:</p> <pre>buf1 = nil collectgarbage()</pre> <p>Key Note: Don't delete the reading buffers by doing</p> <pre>for name in dmm.buffer.catalog() do name = nil end</pre> <p>This will not delete the reading buffers from the system but make the system appear locked up and an abort will need to be done to stop the command from running by pressing the EXIT key on front panel. This occurs because <code>name</code> is a string type variable and not a reading buffer type.</p>

dmm.buffer.catalog()	
Also see	dmm.buffer.info (on page 13-116)
Example	To print all user-created local reading buffers in the system: <pre>for name in dmm.buffer.catalog() do print(name) end</pre>
dmm.buffer.info()	
Function	Returns the size and capacity of the reading buffer parameter.
Usage	<code>size, capacity = dmm.buffer.info(buffer_name)</code> buffer_name : String representing the reading buffer name to query for size and capacity size : Number representing the n attribute of reading buffer parameter capacity : Number representing the capacity attribute of reading buffer parameter
Remarks	This function uses the specified reading buffer input parameter name to find the corresponding size and capacity to return. Use this function with the <code>dmm.buffer.catalog()</code> function to output the size and capacity for all reading buffers in the system.
Also see	dmm.buffer.catalog() (on page 13-115)

dmm.buffer.info()	
Example	<p>Assume the system has the following reading buffers created: buf1, buf2, buf3, buf4, and buf5.</p> <p>Now, query the system for the size and capacity of each reading buffer without formatting the results.</p> <pre>for n in dmm.buffer.catalog() do print(dmm.buffer.info(n)) end</pre> <p>Output results:</p> <pre>0.000000000e+000 2.000000000e+003 0.000000000e+000 4.000000000e+003 0.000000000e+000 5.000000000e+003 0.000000000e+000 3.000000000e+003 0.000000000e+000 1.000000000e+003</pre> <p>Now, to query the system for the name, size, and capacity of each reading buffer while formatting the results:</p> <pre>for n in dmm.buffer.catalog() do size, cap = dmm.buffer.info(n) print(n, 'size = ' .. size, 'capacity = ' .. cap) end</pre> <p>Output results:</p> <pre>buf2 size = 0 capacity = 2000 buf4 size = 0 capacity = 4000 buf5 size = 0 capacity = 5000 buf3 size = 0 capacity = 3000 buf1 size = 0 capacity = 1000</pre>

dmm.buffer.maxcapacity	
Attribute	Indicates the overall max capacity for reading buffers in the system.
Usage	<pre>maxcap = dmm.buffer.maxcapacity</pre> <p>maxcap: Number representing the overall max capacity for reading buffers</p>

dmm.buffer.maxcapacity	
Remarks	Use this attribute to determine what the system maximum capacity for reading buffer storage is. This value represents the total system reading buffer storage size. A single reading buffer may be created (<code>dmm.makebuffer()</code> (on page 7-8) with this as its size or several reading buffers may be created in the system that are smaller in size. However, the sum total of all reading buffer sizes in the system can't exceed this maximum. For example, <pre>print(dmm.buffer.maxcapacity) -> 6.500000000e+005</pre> So we have 650,000 readings as our max capacity.
Also see	<code>dmm.buffer.usedcapacity</code> (on page 13-118) <code>dmm.buffer.info</code> (on page 13-116)
Example	To read the maximum reading buffer capacity for the system: <pre>MaxBuffCap = dmm.buffer.maxcapacity</pre>

dmm.buffer.usedcapacity	
Attribute	Indicates how much of the maximum capacity for reading buffers in the system is used.
Usage	<pre>usedcap = dmm.buffer.usedcapacity</pre> usedcap: Number representing current used capacity for reading buffers in system
Remarks	Use this attribute to determine how much of the system maximum capacity for reading buffer storage is used. This value represents the sum total capacity of all reading buffers in the system. For example, assume the following commands have been executed: <pre>buf1 = dmm.makebuffer(300000) buf2 = dmm.makebuffer(300000)</pre> Therefore: <pre>print(dmm.buffer.usedcapacity) -> 6.000000000e+005 print(dmm.buffer.maxcapacity - dmm.buffer.usedcapacity) -> 5.000000000e+004</pre> This shows that we have 50,000 available for creating additional reading buffers.
Also see	<code>dmm.buffer.maxcapacity</code> (on page 13-117) <code>dmm.buffer.info</code> (on page 13-116)
Example	To read the used reading buffer capacity for the system: <pre>UsedBuffCap = dmm.buffer.usedcapacity</pre>

dmm.calibration.ac()	
Function	Signals the desired AC calibration step on the DMM.
Usage	<p><code>dmm.calibration.ac(step, value)</code></p> <p>step: Represents the AC calibration step to perform</p> <p>value Represents the associated value for this functions step. This is an optional parameter. Only use if calibration step has a value associated. If no value is needed, use <code>dmm.calibration.ac(step)</code>.</p>
Remarks	Use this command to indicate the desired AC calibration step to perform on the DMM. This command will generate an error if the step is out of sequence, does not exist, or the calibration is locked. An error will be generated if the calibration step does not complete successfully or if the value passed is invalid for the step, out of range, or not needed.
Details	<p>AC volts calibration</p> <p><code>dmm.calibration.ac(1)</code> ' AC cal step 1 (10mV, 1kHz step)</p> <p><code>dmm.calibration.ac(2)</code> ' AC cal step 2 (100mV, 1kHz step)</p> <p><code>dmm.calibration.ac(3)</code> ' AC cal step 3 (100mV, 50kHz step)</p> <p><code>dmm.calibration.ac(4)</code> ' AC cal step 4 (1V, 1kHz step)</p> <p><code>dmm.calibration.ac(5)</code> ' AC cal step 5 (1V, 50kHz step)</p> <p><code>dmm.calibration.ac(6)</code> ' AC cal step 6 (10V, 1kHz step)</p> <p><code>dmm.calibration.ac(7)</code> ' AC cal step 7 (10V, 50kHz step)</p> <p><code>dmm.calibration.ac(8)</code> ' AC cal step 8 (100V, 1kHz step)</p> <p><code>dmm.calibration.ac(9)</code> ' AC cal step 9 (100V, 50kHz step)</p> <p><code>dmm.calibration.ac(10)</code> ' AC cal step 10 (300V, 1kHz step)</p>

dmm.calibration.ac()	
Details, continued	<p>AC current calibration</p> <pre>dmm.calibration.ac(11) ' AC cal step 11 (100uA 1kHz step) dmm.calibration.ac(12) ' AC cal step 12 (1mA 1kHz step) dmm.calibration.ac(13) ' AC cal step 13 (10mA 1kHz step) dmm.calibration.ac(14) ' AC cal step 14 (100mA 1kHz step) dmm.calibration.ac(15) ' AC cal step 15 (1A 1kHz step) dmm.calibration.ac(16) ' AC cal step 16 (2A 1kHz step)</pre>
Example	<p>To perform AC calibration step 1 after unlocking calibration:</p> <pre>dmm.calibration.ac(1)</pre>

dmm.calibration.dc()	
Function	Signals the desired DC calibration step on the DMM.
Usage	<pre>dmm.calibration.dc(step, value)</pre> <p>step: Represents the DC calibration step to perform</p> <p>value: Represents the associated value for the specified step. This is an optional parameter; only use if step has a value associated. If no value is needed, use <code>dmm.calibration.dc(step)</code>.</p>
Remarks	Use this command to indicate the desired DC calibration step to perform on the DMM. This command will generate an error if the step is out of sequence, does not exist, or the calibration is locked. An error will be generated if the calibration step does not complete successfully or if the value passed is invalid for the step (out of range or not needed).

dmm.calibration.dc()	
Details	<p>DC volts calibration</p> <p>dmm.calibration.dc(1) \ DC cal step 1 (4 wire short circuit step)</p> <p>dmm.calibration.dc(2) \ DC cal step 2 (open circuit step)</p> <p>dmm.calibration.dc(3, <value>) \ DC cal step 3 (+10V step, 9 < value < 11)</p> <p>dmm.calibration.dc(4, <value>) \ DC cal step 4 (-10V step, -11 < value < -9)</p> <p>dmm.calibration.dc(5, <value>) \ DC cal step 5 (+100V step, 90 < value < 110)</p> <p>Resistance calibration</p> <p>dmm.calibration.dc(6, <value>) \ DC cal step 6 (100 ohm step, 90 < value < 110)</p> <p>dmm.calibration.dc(7, <value>) \ DC cal step 7 (10k ohm step, 9k < value < 11k)</p> <p>dmm.calibration.dc(8, <value>) \ DC cal step 8 (100k ohm step, 90k < value < 110k)</p> <p>dmm.calibration.dc(9, <value>) \ DC cal step 9 (1M ohm step, .9M < value < 1.1M)</p> <p>DC current calibration</p> <p>dmm.calibration.dc(10, <value>) \ DC cal step 10 (100uA step, 90u < value < 110u)</p> <p>dmm.calibration.dc(11, <value>) \ DC cal step 11 (1mA step, .9m < value < 1.1m)</p> <p>dmm.calibration.dc(12, <value>) \ DC cal step 12 (10mA step, 9m < value < 11m)</p> <p>dmm.calibration.dc(13, <value>) \ DC cal step 13 (100mA step, 90m < value < 110m)</p> <p>dmm.calibration.dc(14, <value>) \ DC cal step 14 (1A step, .9 < value < 1.1)</p>
Example	<p>To perform DC calibration step 2 after unlocking calibration: dmm.calibration.dc(2)</p> <p>To perform DC calibration step 3 with a value of 10 after unlocking calibration: dmm.calibration.dc(3, 10)</p>

dmm.calibration.lock()	
Function	Locks calibration.
Usage	<code>dmm.calibration.lock()</code>
Remarks	Use this command to lock an unlocked calibration. Once locked, calibration will need to be unlocked to be performed again. An error will be generated if this command is issued when calibration is already locked. This function locks calibration but does not save calibration data. Calibration data will be lost if it is not saved before locking.
Also see	dmm.calibration.save() (on page 13-122)
Example	To lock calibration: <code>dmm.calibration.lock()</code>

dmm.calibration.password	
Attribute	Sets the password to unlock calibration.
Usage	<code>dmm.calibration.password = password</code> password: A string representing the valid password to unlock calibration.
Remarks	Use this attribute to set the password to unlock calibration. Make note of the password because there is no command to query for the password once set on the system. This command will generate an error if calibration is locked or if the password string length is greater than 10 characters. This item can only be set when calibration is unlocked.
Example	To set the password to "MyUnlock" after unlocking calibration with the saved password: <code>dmm.calibration.password = "MyUnlock"</code>

dmm.calibration.save()	
Function	Saves calibration data.
Usage	<code>dmm.calibration.save()</code>
Remarks	Use this command to save calibration data after performing a calibration. This command needs to be received before locking calibration, otherwise the data will be lost. This command will save the calibration constants, adjustment date, and increase the adjustment count by 1. An error will be generated if this command is issued when calibration is already locked. This command will save the calibration data with its present values regardless of errors in the data. The user should not issue a save unless the calibration procedure was performed with no errors. If no calibration date was specified (see dmm.adjustment.date (on page 13-109)), the date will be auto generated based on the system date.
Also see	dmm.calibration.lock() (on page 13-121) dmm.adjustment.date (on page 13-109)
Example	To save calibration data: <code>dmm.calibration.save()</code>

dmm.calibration.unlock()	
Function	Unlocks calibration.
Usage	<code>dmm.calibration.unlock(password)</code> password: A string representing the password to unlock calibration.
Remarks	Use this command to unlock calibration (if locked). An error will be generated if the password does not match the one saved. The default password from the factory is "KI003706". This may be changed with dmm.calibration.password (on page 13-122).
Also see	dmm.calibration.password (on page 13-122)
Example	To unlock calibration using the default password: <code>dmm.calibration.unlock("KI003706")</code>

dmm.calibration.verifydate	
Attribute	Set or queries the calibration verification date in UTC format (number of seconds since January 1, 1970).
Usage	To query the calibration verify date: <code>CalDate = dmm.calibration.verifydate</code> CalDate: Represents the number of seconds since January 1, 1970. To set the calibration verification date based on the current date of the system: <code>dmm.calibration.verifydate = os.time()</code> To set the calibration verification date as July 4, 2007: <code>dmm.calibration.verifydate = os.time({year=2007, month=7, day = 4})</code>
Remarks	This attribute sets and gets the calibration verification date of the DMM in UTC format. See Lua documentation for formatting options with <code>os.date</code> . This item can only be set when calibration is unlocked.
Example	NOTE Example assumes the set date is July 4, 2007. To query calibration verification date and format the response as mm/dd/yyyy: <code>print(os.date("%m/%d/%Y", dmm.calibration.verifydate))</code> → 07/04/2007 To query calibration verification date and format the response as mm/dd/yy: <code>print(os.date("%x", dmm.calibration.verifydate))</code> → 07/04/07

dmm.close()	
Function	Closes the specified channel or channel pattern in preparation for a DMM measurement.
Usage	<code>dmm.close(<ch_list>)</code> ch_list: string listing the channel or channel pattern to close.

dmm.close()	
Remarks	<p>The actions associate with this function includes:</p> <ul style="list-style-type: none"> • Opens previously closed channels if they interfere with measurement including analog backplane relays 1, 2, and common-side ohms on all slots, if needed. The opening and closing of channels mimics that of channel.exclusiveslotclose() (on page 13-43). Therefore, when using a for-loop with dmm.close() (on page 13-123) command, the last channel on each slot will be closed at the end of the for loop execution. To have additional analog backplane relays 3 through 6 closed, use them on an alternate slot or if need to be on same slot then create a channel pattern. To have additional channels closed, then use patterns. Using patterns, you must specify all items to close, including analog backplane relays 1 and 2. With patterns, there is no auto manipulation of analog backplane relays 1 and 2 like there is with channels. • Any amp channels will open as well. If there is a need to have multiple amp channels closed at a time then create a channel pattern. • Next, this command will close the associated channels and analog backplane relays which include analog backplane relay 1 and 2 as needed based on configuration associated with channel (see dmm.getconfig() (on page 13-139)) Analog backplane relays specified by channel.setbackplane() (on page 13-70) are not used. • This command will configure the DMM based on the configuration associated with the channel or channel pattern being closed (see dmm.getconfig() (on page 13-139)). If the configuration is a default name, the function of that configuration will be reset back to factory default settings. User must create a unique DMM configuration to avoid using factory default settings when assigning to a channel (see dmm.configure.set() (on page 13-129) and dmm.setconfig() (on page 13-168) commands.)
Details	<p>An error will be generated if:</p> <ul style="list-style-type: none"> • There is a syntax error in parameter string. • An empty parameter string is specified. • The specified channel or channel pattern is invalid. • The channel number does not exist for slot specified. • The slot is empty. • The channel pattern does not exist. • A forbidden item is specified. • The specified channel does not support being closed (like a digital I/O channel). • More than 1 channel or channel pattern specified. • The channel is paired with another bank for a multi-wire application. • The channel is an analog backplane relay. • The channel's configuration is 'nofunction'. <p>Once an error is detected, the command stops processing. Channels will open and close, and the DMM will be reconfigured only if no error is detected. Otherwise, channels and DMM remain unchanged.</p> <p>This command allows you to separate the closing of channels from measuring. Therefore, you may execute any number of commands between the close and measure commands to satisfy your application needs.</p>

dmm.close()	
Also see	channel.exclusiveslotclose() (on page 13-43) channel.getclose() (on page 13-46) channel.getstate() (on page 13-56) dmm.open() (on page 13-154)
Example	<p>To close Channel 3 on Slot 3 and prepare the DMM for measuring temperature at 'mytemperature' settings:</p> <pre>dmm.setconfig('3003', 'mytemperature') dmm.close('3003')</pre> <p>To close a channel pattern called mychans and prepare DMM for measuring 'dcvolts' at factory default settings:</p> <pre>dmm.setconfig('mychans', 'dcvolts') dmm.close('mychans')</pre>
dmm.configure.catalog()	
Function	Creates an iterator for the user-created DMM configurations.
Usage	<code>for name in dmm.configure.catalog() do ... end</code>
Remarks	Accessing the catalog for user DMM configurations allows the user to print or delete all configurations in volatile memory. The entries will be enumerated in no particular order. This will only list user-created DMM configurations; it does not list the factory default configurations.
Also see	dmm.configure.delete() (on page 13-125) dmm.configure.query() (on page 13-126) dmm.configure.recall() (on page 13-128) dmm.configure.set() (on page 13-129)
Example	<p>To delete all user-created DMM configurations from volatile memory:</p> <pre>for name in dmm.configure.catalog() do dmm.configure.delete(name) end</pre> <p>To print all user-created DMM configurations in volatile memory:</p> <pre>for name in dmm.configure.catalog() do print(name) end</pre>
dmm.configure.delete()	
Function	Deletes the specified user created DMM configuration from memory.
Usage	<code>dmm.configure.delete(name)</code> name: String containing the name of the DMM configuration to delete

dmm.configure.delete()	
Remarks	The function will delete the specified DMM configuration from memory. An error will be generated if the name specified does not exist as a user configuration. After executing this command, the specified name will no longer exist as a valid configuration. Deleting an existing DMM configuration invalidates an existing scan list.
Also see	dmm.configure.catalog() (on page 13-125) dmm.configure.set() (on page 13-129) dmm.configure.query() (on page 13-126) dmm.configure.recall() (on page 13-128)
Example	To delete a user configuration called myDCV: <pre>dmm.configure.delete("myDCV")</pre>

dmm.configure.query()	
Function	Provides a list of all of the pertinent DMM settings associated with a configuration.
Usage	<pre>config = dmm.configure.query(myconfig, myseparator)</pre> <p>myconfig: A string with the name for the DMM configuration being listed.</p> <p>myseparator: String representing the two-character separator between items. This is an optional parameter. Default value: , (comma-space delimited)</p> <p>config: Output string representing the DMM attribute settings of myconfig.</p>

dmm.configure.query()	
Remarks	<p>This function will list the settings contained within the specified configuration, <code>myconfig</code>, along with the corresponding DMM attributes contained within that configuration. A nil response is generated if the specified configuration does not exist, along with an error message stating the referenced name does not exist. The second parameter, <code>myseparator</code>, is optional. If not specified, the attributes are comma delimited within the config response. However, if specified, then the attributes will be delimited by this two-character separator. If more than two characters are specified, an error message (string is too long) is generated. To query the factory default settings for a function, use the desired function for the <code>myconfig</code> parameter. Valid default functions are:</p> <ul style="list-style-type: none"> • "dcvolts" • "acvolts" • "dcurrent" • "accurrent" • "twowireohms" • "fourwireohms" • "temperature" • "frequency" • "period" • "continuity" • "commonsideohms" • "nofunction" <p>To query for the settings associated with the active function, call this function with the <code>myconfig</code> parameter set to "active".</p>
Also see	<p>dmm.configure.catalog() (on page 13-125)</p> <p>dmm.configure.set() (on page 13-129)</p> <p>dmm.configure.delete() (on page 13-125)</p> <p>dmm.configure.recall() (on page 13-128)</p>

dmm.configure.query()	
Example	<p>To see the DMM attributes within MyDcv separated by commas:</p> <pre>MyDcvItems = dmm.configure.query("MyDcv") print(MyDcvItems)</pre> <p>To see the DMM attributes within MyDcv separated by newlines:</p> <pre>MyDcvItems = dmm.configure.query("MyDcv", "\n") print(MyDcvItems)</pre> <p>To see the factory default settings for DC volts separated by newlines:</p> <pre>FactoryDCV = dmm.configure.query("dcvolts", "\n") print(FactoryDCV)</pre> <p>To see the DMM attributes for the active function separated by newlines:</p> <pre>ActiveFunc = dmm.configure.query("active", "\n") print(ActiveFunc)</pre>
dmm.configure.recall()	
Function	Recalls a user or factory DMM configuration.
Usage	<pre>dmm.configure.recall(config)</pre> <p>config: A string representing the name of the DMM configuration to recall.</p>

dmm.configure.recall()	
Remarks	<p>This command will recall a saved DMM configuration from memory. The DMM attributes associated with that configuration will be updated to reflect the settings in the configuration. The ones not pertinent will not be changed. The function associated with the configuration will become the active one. The previous values of those being updated will be lost. The specified DMM configuration to recall may be a factory one or user one.</p> <p>To recall a factory configuration, send the function name as the config value. Valid factory configurations are:</p> <ul style="list-style-type: none"> • "dcvolts" • "acvolts" • "dcurrent" • "accurrent" • "twowireohms" • "fourwireohms" • "temperature" • "frequency" • "period" • "continuity" • "commonsideohms" • "nofunction" <p>Recalling a factory configuration also changes the corresponding DMM function (dmm.func) and resets that function's attributes to their default values. Settings not pertaining to the function will remain unchanged.</p> <p>An error is generated if the specified configuration does not exist in memory.</p>
Also see	<p>dmm.configure.set() (on page 13-129)</p> <p>dmm.configure.delete() (on page 13-125)</p> <p>dmm.configure.query() (on page 13-126)</p> <p>dmm.func (on page 13-137)</p>
Example	<p>To recall a DMM configuration call "mydmm":</p> <pre>dmm.configure.recall("mydmm")</pre> <p>To recall factory default settings for "temperature":</p> <pre>dmm.configure.recall("temperature")</pre>

dmm.configure.set()	
Function	Creates a DMM configuration with the pertinent attributes based on the selected function and associates it with the specified name.
Usage	<pre>dmm.configure.set(config)</pre> <p>config: A string with the name for the DMM configuration being created.</p>

dmm.configure.set()	
Remarks	<p>If the configuration specified is being used for an existing DMM configuration, then that configuration will be overwritten with the new configuration settings, if no errors occur. The previous configuration associated with config will be lost. All channels that have this configuration as their associated DMM configurations now will start using the new attribute settings.</p> <p>This command creates a copy of the selected function along with its pertinent settings to volatile memory to be recalled later when desired. If a DMM setting is not pertinent it will not be stored. Therefore, creating a configuration with dcvolts set as the function will not store the temperature settings. A DMM configuration has only one function associate with it and enough settings information to support that function and no more.</p> <p>An error will be generated if the name being used to create a configuration corresponds to a factory default configuration.</p> <p>If the specified name already exists then that configuration will be overwritten with the new information and the previous configuration will be lost.</p> <p>Executing a <code>dmm.configure.set</code> on an existing DMM configuration will invalidate an existing scan list (the DMM configuration may or may not be used in the current scan list). If the configuration being created does not already exist, the scan list will not be invalidated.</p> <p>The maximum number of characters for a named configuration is 30. An error will be generated if the number of characters in config parameter is greater than 30.</p>
Details	These are not persistent through a power cycle. User DMM configurations are part of the data associated with a saved setup.
Also see	<p>dmm.configure.catalog() (on page 13-125)</p> <p>dmm.configure.delete() (on page 13-125)</p> <p>dmm.configure.query() (on page 13-126)</p> <p>dmm.configure.recall() (on page 13-128)</p>
Example	<p>To create a DMM configuration as "mydmm":</p> <pre>dmm.configure.set("mydmm")</pre>

dmm.connect	
Attribute	Indicates how the DMM relays should be connected to the analog backplane.

dmm.connect	
Usage	<p>To read the DMM relay connection setting:</p> <pre>value = dmm.connect</pre> <p>value: Represents the present DMM relay connection setting</p> <p>To write the DMM relay connection setting:</p> <pre>dmm.connect = value</pre> <p>value: Represents the desired DMM relay connection setting</p> <p>where value is:</p> <ul style="list-style-type: none"> • dmm.CONNECT_NONE or 0 to have no relays connected • dmm.CONNECT_ALL or 7 to have all relays connected (default value) • dmm.CONNECT_TWO_WIRE or 1 to have 2-wire relay connected • dmm.CONNECT_FOUR_WIRE or 3 to have 2-wire & sense relays connected • dmm.CONNECT_TWO_WIRE_AMPS or 5 to 2-wire & amps relay connected • dmm.CONNECT_AMPS or 4 to have amps relay connected
Remarks	<p>Use the value setting, as indicated in the usage section to indicate, which of the DMM relays you want connected to the backplane. The relays are bitmapped into the lower 3 bits of the value where bit 0, a value of 1, represents the 2-wire relay. Bit 1, a value of 2, represents the sense relay and bit 3, a value of 4, represents the amp relay. By setting the appropriate bit to a 1, then that relay is closed. Likewise, setting it to zero, opens that relay.</p> <p>An error will be generated only if the sense relay bit is set to a 1 and if the sense relay with amps is selected. These two settings correspond to a value of 2 or 6, respectively. Also, an error will be generated for a value less than 0 or greater than 7.</p> <p>Default setting is dmm.CONNECT_ALL.</p>
Details	Use of this command is not recommended with the exception of special cases. The default setting should handle most applications.
Example	<p>To connect the DMM 2-wire and amp relays to the analog backplane:</p> <pre>dmm.connect = dmm.CONNECT_TWO_WIRE_AMPS</pre>

dmm.dbreference	
Attribute	Indicates the DB reference setting for the DMM in volts.
Usage	<p>To read the DB reference:</p> <pre>value = dmm.dbreference</pre> <p>value: Represents the present DB reference setting in volts</p> <p>To write the DB reference:</p> <pre>dmm.dbreference = value</pre> <p>value: Represents the desired DB reference in volts (1e-7 to 1000)</p>

dmm.dbreference	
Remarks	<p>This is the DB reference setting for the DMM and it applies to the selected function as indicated by <code>dmm.func</code>. Querying this setting when the selected function does not support it will cause nil to be returned.</p> <p>Command only applies when <code>dmm.func = "dcvolts"</code> or <code>"acvolts"</code>. For all other functions an error will be generated if this command is received. Also, an error will be generated if the value is out of range.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the DB reference setting for that function.</p>
Example	<p>To set the DB reference to 5 volts for DC volts:</p> <pre>dmm.func = "dcvolts" dmm.dbreference = 5</pre>
dmm.detectorbandwidth	
Attribute	Indicates the detector bandwidth setting for the DMM in Hertz.
Usage	<p>To read the detector bandwidth:</p> <pre>value = dmm.detectorbandwidth</pre> <p>value: Represents the present detector bandwidth setting in Hertz</p> <p>To write the detector bandwidth:</p> <pre>dmm.detectorbandwidth = value</pre> <p>value: Represents the desired detector bandwidth in Hertz (3 to 300). Values less than 30, the parameter is adjusted to 3. Values between 3 and 300, the parameter is adjusted to 30. For values greater than 300, the parameter is adjusted to 300.</p>
Remarks	<p>This is the AC detector bandwidth setting for the DMM and it applies to the selected function as indicated by <code>dmm.func</code>. Querying the aperture when the selected function does not have a detector bandwidth setting associated with it will cause nil to be returned.</p> <p>Command only applies when <code>dmm.func = "acvolts"</code> or <code>"accurrent"</code>. For all other functions an error will be generated if this command is received. Also, an error will be generated if the value is out of range. The default setting for both functions is 300.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the detector bandwidth setting for that function.</p> <p>When trying to set the aperture for functions with dmm.aperture (on page 13-110) attribute, if the detector bandwidth setting is 30 or less, an error message will be generated.</p>
Example	<p>To set the detector bandwidth to 100 Hertz for AC volts:</p> <pre>dmm.func = "acvolts" dmm.detectorbandwidth = 100 -- this gets adjusted to 30 print(dmm.detectorbandwidth) --> "30"</pre>

dmm.displaydigits	
Attribute	Indicates the display digits setting for the selected DMM function.
Usage	<p>To read the display digits setting:</p> <pre>value = dmm.displaydigits</pre> <p>value: Represents the present display digits setting.</p> <p>To write the display digits setting:</p> <pre>dmm.displaydigits = value</pre> <p>value: Represents the desired display digits setting. Set to one of the following:</p> <p>dmm.DIGITS_7_5 or 7 to enable 7.5 display digits dmm.DIGITS_6_5 or 6 to enable 6.5 display digits dmm.DIGITS_5_5 or 5 to enable 5.5 display digits dmm.DIGITS_4_5 or 4 to enable 4.5 display digits dmm.DIGITS_3_5 or 3 to enable 3.5 display digits</p>
Remarks	<p>This is the display digits setting for the DMM and it applies to the selected function as indicated by dmm.func. Querying the setting when the selected function doesn't support it will cause nil to be returned.</p> <p>An error will be generated if the value is invalid.</p> <p>Changing functions with dmm.func has no effect on this setting.</p> <p>The factory default is 6 because on "dcvolts" and dmm.reset value is dependent on function resetting.</p>
Example	<p>To enable display digits to 7.5 for dcvolts:</p> <pre>dmm.func = "dcvolts"</pre> <pre>dmm.displaydigits = dmm.DIGITS_7_5</pre>

dmm.drycircuit	
Attribute	Indicates the dry circuit setting for the selected DMM function.
Usage	<p>To read the dry circuit:</p> <pre>value = dmm.drycircuit</pre> <p>value: Represents the present dry circuit setting</p> <p>To write the dry circuit:</p> <pre>dmm.drycircuit = value</pre> <p>value Represents the desired dry circuit setting. Use one of the following:</p> <p>dmm.ON or 1 to enable dry circuit. dmm.OFF or 0 to disable dry circuit</p>

dmm.drycircuit	
Remarks	<p>This is the dry circuit setting for the DMM and it applies to the selected function as indicated by <code>dmm.func</code>. Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>This command only applies when <code>dmm.func = "fourwireohms"</code> or <code>"commonsideohms"</code>. All other function settings will generate an error if the command is received. Also, an error will be generated if the value is invalid.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the dry circuit setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is <code>dmm.OFF</code>.</p>
Example	<p>To enable dry circuit for 4-wire ohms:</p> <pre>dmm.func = "fourwireohms" dmm.drycircuit = dmm.ON</pre>
dmm.filter.count	
Attribute	Indicates the filter count setting for the selected DMM function.
Usage	<p>To read the filter count:</p> <pre>value = dmm.filter.count</pre> <p>value: Represents the present filter count setting</p> <p>To write the filter count:</p> <pre>dmm.filter.count = value</pre> <p>value: Represents the desired filter count setting from 1 to 100</p>
Remarks	<p>This is the filter count setting for the DMM and it applies to the selected function as indicated by <code>dmm.func</code>. Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>This attribute indicates the number of measured readings to yield one filtered measurements when filtered measurements are enabled.</p> <p>The command will generate an error when <code>dmm.func = "frequency"</code>, <code>"period"</code>, <code>"continuity"</code> or <code>"nofunction"</code>. Also, an error will be generated if the value is out of range.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the filter count setting for that function.</p> <p>The dmm.reset() (on page 13-161) function sets the filter count to 10.</p>
Also see	<p>dmm.filter.enable (on page 13-134)</p> <p>dmm.filter.type (on page 13-135)</p>
Example	<p>To set the filter count for 2-wire ohms to 5:</p> <pre>dmm.func = "twowireohms" dmm.filter.count = 5</pre>

dmm.filter.enable	
Attribute	Enables or disables filtered measurements for the selected DMM function.
Usage	<p>To read the filter enable:</p> <pre>value = dmm.filter.enable</pre> <p>value: Represents the present filter enable setting</p> <p>To write the filter enable:</p> <pre>dmm.filter.enable = value</pre> <p>value: Represents the desired filter enable setting. Use one of the following:</p> <ul style="list-style-type: none"> • dmm.ON or 1 to enable filter measurements • dmm.OFF or 0 to disable filter measurements
Remarks	<p>This is the filter enable setting for the DMM and it applies to the selected function as indicated by <code>dmm.func</code>. Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>This attribute indicates whether filtered measurements are enabled (or not).</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the filter enable setting for that function.</p> <p>The dmm.reset() (on page 13-161) function disables the filter.</p>
Also see	dmm.filter.count (on page 13-134), dmm.filter.type (on page 13-135), dmm.filter.window (on page 13-136)
Example	<p>To enable the filter for 2-wire ohms:</p> <pre>dmm.func = "twowireohms"</pre> <pre>dmm.filter.enable = dmm.ON</pre>

dmm.filter.type	
Attribute	Indicates the filter type for the DMM measurements on selected DMM functions.
Usage	<p>To read the filter type:</p> <pre>value = dmm.filter.type</pre> <p>value: Represents the present filter type setting</p> <p>To write the filter type:</p> <pre>dmm.filter.type = value</pre> <p>value: Represents the desired filter type for measurements. Use:</p> <ul style="list-style-type: none"> • dmm.FILTER_MOVING_AVG or 0 for moving average filter • dmm.FILTER_REPEAT_AVG or 1 for repeat filtering

dmm.filter.type	
Remarks	<p>This is the filter type setting for the DMM and it applies to the selected function as indicated by dmm.func (on page 13-137). Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>There are two averaging filter types to choose from: Repeating and moving. For the repeating (which is 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.</p> <p>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.</p> <p>The command will generate an error when <code>dmm.func = "frequency", "period", "continuity" or "nofunction"</code>. Also, an error will be generated if the value is invalid.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the filter type setting for that function.</p> <p>The dmm.reset() (on page 13-161) function selects the repeat filter.</p>
Also see	dmm.filter.count (on page 13-134), dmm.filter.enable (on page 13-134), dmm.filter.window (on page 13-136)
Example	<p>To set the filter type for 2-wire ohms to moving average:</p> <pre>dmm.func = "twowireohms" dmm.filter.type = dmm.FILTER_MOVING_AVG</pre>

dmm.filter.window	
Attribute	Indicates the filter window for the DMM measurements
Usage	<p>To read the filter window:</p> <pre>value = dmm.filter.window</pre> <p>value: Represents the present filter window setting</p> <p>To write the filter window:</p> <pre>dmm.filter.window = value</pre> <p>value: Represents the desired filter window for measurements. Use a value between 0 and 10 to indicate percent of range.</p>
Remarks	<p>This is the filter window setting for the DMM and it applies to the selected function as indicated by dmm.func (on page 13-137). Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>This attribute indicates the filter window to use in percent of range.</p> <p>The command will generate an error when <code>dmm.func = "frequency", "period", "continuity" or "nofunction"</code>. Also, an error will be generated if the value is out of range.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the filter window setting for that function.</p> <p>The dmm.reset() (on page 13-161) function selects a filter window of 0.1.</p>

dmm.filter.window	
Also see	dmm.filter.enable (on page 13-134), dmm.filter.count (on page 13-134), dmm.filter.type (on page 13-135).
Example	To set the filter window for 2-wire ohms to 0.25: <pre>dmm.func = "twowireohms" dmm.filter.window = 0.25</pre>
dmm.fourrtd	
Attribute	Indicates the type of 4-wire RTD being used.
Usage	To read the 4-wire RTD type: <pre>value = dmm.fourrtd</pre> <p>value: Represents the present type for 4-wire RTD</p> <p>To write the 4-wire RTD type: <pre>dmm.fourrtd = value</pre> <p>value: Represents the desired type for 4-wire RTD. Use one of the following for values:</p> <ul style="list-style-type: none"> • dmm.RTD_PT100 or 0 for type PT100 • dmm.RTD_D100 or 1 for type D100 • dmm.RTD_F100 or 2 for type F100 • dmm.RTD_PT385 or 3 for type PT385 • dmm.RTD_PT3916 or 4 for type PT3916 • dmm.RTD_USER or 5 for user specified type </p>
Remarks	This attribute is only valid when <code>dmm.func = "temperature"</code> . All other configurations generate an error and return nil when queried. When the function is temperature, the 4-wire RTD is only used when the transducer type is 4-wire RTD (see dmm.transducer (on page 13-173)). For all other transducer types, the setting will be updated but ignored until the transducer type is set for 4-wire RTD.
	Changing functions with dmm.func (on page 13-137) will reflect the 4-wire RTD setting for that function.
	The dmm.reset() (on page 13-161) function will set this attribute to <code>dmm.RTD_PT100</code> .
Example	To set the type of 4-wire RTD for PT3916: <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_FOURRTD dmm.fourrtd = dmm.RTD_PT3916</pre>

dmm.func	
Attribute	Sets or indicates the selected function for the DMM.
Usage	<p>To read the function: <code>myfunc = dmm.func</code></p> <p>myfunc: string indicating the presently selected DMM function</p> <p>To write the function: <code>dmm.func = myfunc</code></p> <p>myfunc: string representing the desired active DMM function</p> <p>Set myfunc to one of the following:</p> <ul style="list-style-type: none"> • "dcvolts" or dmm.DC_VOLTS • "acvolts", or dmm.AC_VOLTS • "dccurrent" or dmm.DC_CURRENT • "accurrent" or dmm.AC_CURRENT • "twowireohms" or dmm.TWO_WIRE_OHMS • "fourwireohms" or dmm.FOUR_WIRE_OHMS • "temperature" or dmm.TEMPERATURE • "frequency" or dmm.FREQUENCY • "period" or dmm.PERIOD • "continuity" or dmm.CONTINUITY • "commonsideohms" or dmm.COMMON_SIDE_OHMS • "nofunction" or dmm.NO_FUNCTION
Remarks	<p>Setting this attribute changes the selected DMM function and indicates how the other DMM attributes are to be processed. It does not modify any attribute of the DMM except the one indicating the selected DMM functionality. However, the DMM gets updated to use the current values for each attribute that pertains to the newly selected function. The current values being used are the ones from when the selected function was last active.</p> <p>An error will be generated if the setting does not match one of the ones specified in usage. If an error is found, no change is made to the function.</p> <p>Default setting is "dcvolts". An error will occur if a user DMM configuration name is used to set the function.</p>

dmm.func	
Details	<p>The DMM has a flat view of settings in terms of commands. However, internally, the DMM maintains settings on per function basis. Therefore, to see a setting for a particular function, you need to change to that function with this command (dmm.func) and then write or read the desired setting. For example, to see the NPLC setting for DC volts:</p> <pre>dmm.func = "dcvolts" dcm_nplc = dmm.nplc</pre> <p>With the DMM internally maintaining settings on a per function basis, you may change to a function and write your desired settings. Next, change to another function and write those desired settings. Now, you may switch back to your original function and those settings will be there. For example:</p> <pre>dmm.func = dmm.DC_VOLTS dmm.nplc = 0.5 dmm.range = 10 dmm.func = "twowireohms" dmm.nplc = 0.1 dmm.range = 100000 dmm.func = "dcvolts" print(dmm.nplc) → 0.5 print(dmm.range) → 10 dmm.func = dmm.TWO_WIRE_OHMS print(dmm.nplc) → 0.1 print(dmm.range) → 100000</pre>
Also see	dmm.configure.recall() (on page 13-128)
Example	<p>To make "temperature" the active DMM function:</p> <pre>dmm.func = "temperature"</pre>

dmm.getconfig()	
Function	Queries for the DMM configurations associated with channel list parameter items.
Usage	<pre>config =dmm.getconfig(<ch_list>)</pre> <p>ch_list: A string indicating channels and/or channel patterns to query.</p> <p>config: A comma-delimited string listing DMM configurations associated with items in ch_list.</p>

dmm.getconfig()	
Remarks	<p>Use this command to query for DMM configurations associated with channels or channel patterns. The response will be a comma-delimited string listing the configurations in the same order as specified in <code>ch_list</code>.</p> <p>The configurations listed in the response indicate how the DMM will be configured when the corresponding channel or channel pattern is closed with the <code>dmm.close()</code> (on page 13-123) function or used in a scan list without an overriding DMM configuration.</p> <p>An error will be generated if:</p> <ul style="list-style-type: none"> • Syntax error exists in parameter string. • An empty parameter string is specified. • A specified channel or channel pattern is invalid. • Channel number does not exist for slot based on installed card. • Slot is empty. • Channel pattern does not exist. • Channel being specified does not support a configuration setting like a digital I/O channel or analog backplane relay. <p>Command processing will stop once an error is detected and a nil response is generated.</p>
Also see	dmm.setconfig() (on page 13-168)
Example	<p>To query channels on Slots 1 and 2:</p> <pre>myconfigs = dmm.getconfig('slot1, slot2')</pre> <p>To query Channels 1 to 10 on Slot 3:</p> <pre>myconfigs = dmm.getconfig('3001:3010')</pre>

dmm.inputdivider	
Attribute	Enables or disables the 10M ohm input divider.
Usage	<p>To read the input divider state:</p> <pre>value = dmm.inputdivider</pre> <p>value: Represents the present input divider</p> <p>To write the input divider state:</p> <pre>dmm.inputdivider = value</pre> <p>value: Represents the desired input divider state.</p> <p>Use one of the following for value:</p> <ul style="list-style-type: none"> • dmm.ON or 1 • dmm.OFF or 0

dmm.inputdivider	
Remarks	<p>This attribute is only valid when <code>dmm.func = "dcvolts"</code>. All other functions generate an error and return nil when queried.</p> <p>Changing functions with <code>dmm.func</code> (on page 13-137) will reflect the 10M ohm input divider for that function.</p> <p>The factory default and <code>dmm.reset()</code> (on page 13-161) function value is <code>dmm.OFF</code>.</p>
Example	<p>To enable the input divider for DC volts:</p> <pre>dmm.func = "dcvolts" dmm.inputdivider = dmm.ON</pre>

dmm.limit[Y].autoclear		where Y = 1 or 2 for limit number
Attribute	Indicates if limit Y should be cleared automatically or not.	
Usage	<p>To read the auto clear setting:</p> <pre>value = dmm.limit[Y].autoclear</pre> <p>To write the auto clear setting:</p> <pre>dmm.limit[Y].autoclear = value</pre> <p>Use one of the following:</p> <ul style="list-style-type: none"> • <code>dmm.ON</code> or 1 to enable auto clear • <code>dmm.OFF</code> or 0 to disable auto clear 	
Remarks	<p>When this attribute is set to <code>dmm.ON</code>, a limit fail condition will track how the measurements are taken. If the measurement failed limit, then the fail indication will be set. If the next measurement passes limit, the failed limit condition clears. Therefore, if scanning or taking a series of measurements with auto cleared enabled for a limit, then the last measurement limit dictates the fail indication for the limit. To know if any of a series of measurements failed the limit, then set the auto clear setting to off. When set to <code>dmm.OFF</code>, a failed indication will not be cleared automatically and will remain set until it is cleared by <code>dmm.limit[Y].clear()</code> (on page 13-141). The auto clear setting affects both the high and low limits of Y.</p>	
Also see	dmm.measure() (on page 13-150)	
Example	<p>Enable auto clear on limit 2:</p> <pre>dmm.limit[2].autoclear = dmm.ON</pre>	

dmm.limit[Y].clear()		where Y = 1 or 2 for limit number
Function	Clears the test results of limit Y.	
Usage	<code>dmm.limit[Y].clear()</code>	
Remarks	This function clears the test results limit.	
Also see	dmm.limit[Y].high.fail (on page 13-142) dmm.limit[Y].low.fail (on page 13-143)	

dmm.limit[Y].clear()		where Y = 1 or 2 for limit number
Example	To clear the test results for both the high and low limit 2: <code>dmm.limit[2].clear()</code>	
dmm.limit[Y].enable		where Y = 1 or 2 for limit number
Attribute	Enable or disable limit Y testing.	
Usage	<p>To read the state of limit Y: <code>value = dmm.limit[Y].enable</code></p> <p>To write the state of limit Y: <code>dmm.limit[Y].enable = value</code></p> <p>Set value to:</p> <ul style="list-style-type: none"> • dmm.ON or 1 to enable limit Y testing • dmm.OFF or 0 to disable limit Y testing 	
Remarks	<p>When this attribute is set to dmm.ON, the limit Y testing will occur on each measurement taken by the DMM whether being requested by the <code>dmm.measure</code> function or being part of a scan sequence. Limit Y testing involves using the low limit value (specified by dmm.limit[Y].low.value (on page 13-144)) and high limit value (specified by dmm.limit[Y].high.value (on page 13-143)). If the measurement value falls outside either of these limits, then the test fails. Else, it passes. To see the test results, use the dmm.limit[Y].low.fail (on page 13-143) and dmm.limit[Y].high.fail (on page 13-142) attributes.</p> <p>To not use limit Y testing, set the attribute to dmm.OFF. The default setting is dmm.OFF. With limits disabled, limit testing is not performed after taking a measurement. Therefore, the status bits are not updated, the fail indication does not get updated and hardware lines are not generated.</p> <p>Limits are invalid when <code>dmm.func = "continuity"</code>. Trying to use the limit commands with this function selected will generate an error.</p> <p>Limits may be enabled with this command, but without assigning the events to a trigger stimulus for a digital I/O line, there will be no hardware indication of limits. See the <code>dmm.limit[Y].high</code> and <code>dmm.limit[Y].low</code> commands (listed in the "Also See" section).</p>	
Also see	dmm.limit[Y].high.value (on page 13-143) dmm.limit[Y].high.fail (on page 13-142) dmm.limit[Y].low.fail (on page 13-143) dmm.limit[Y].low.value (on page 13-144)	
Example	Enable limit 2 testing: <code>dmm.limit[2].enable = dmm.ON</code>	

dmm.limit[Y].high.fail		where Y = 1 to 2 for limit number
Attribute	Query for the high test results of limit Y.	
Usage	To read the high fail indication of limit Y: value = dmm.limit[Y].high.fail	
Remarks	<p>This attribute returns the results of high limit Y testing:</p> <ul style="list-style-type: none"> • 0 indicates test passed – measurement within the high limit • 1 indicates test failed – measurement has exceeded high limit <p>A failed indication does indicate the high limit caused the failure. You may read the measurement event register of the status model to see fail indication as well. This response only has meaning if the limit's auto clear setting is disabled and the limit is enabled. Otherwise, it will indicate the results of the last limit test when enabled.</p>	
Example	To query the test results of high limit 2: results = dmm.limit[2].high.fail	
dmm.limit[Y].high.value		where Y = 1 or 2 for limit number
Attribute	Indicates the high limit value for limit Y.	
Usage	<p>To read the high value of limit Y: value = dmm.limit[Y].high.value</p> <p>To write the high value of limit Y: dmm.limit[Y].high.value = value</p> <p>value: Valid range of -4294967295 to +4294967295</p>	
Remarks	Use this attribute to specify or query the high limit value of limit Y. When limit Y testing is enabled (dmm.limit[Y].enable (on page 13-142)), the measurement value must be greater than this value to avoid causing a fail indication for the limit. The default value is 1 for limit 1 and 2 for limit 2. You may set or get the value regardless if the limit is set to a digio trigger stimulus (digio.trigger[N].stimulus = dmm.trigger.EVENT_LIMIT1_HIGH or dmm.trigger.EVENT_LIMIT2_HIGH).	
Also see	dmm.limit[Y].high.fail (on page 13-142) dmm.limit[Y].low.value (on page 13-144)	
Example	To set high limit of limit 2 to 5: dmm.limit[2].high.value = 5	
dmm.limit[Y].low.fail		where Y = 1 or 2 for limit number
Attribute	Query for the low test results of limit Y.	
Usage	To read the low fail indication of limit Y: value = dmm.limit[Y].low.fail	

dmm.limit[Y].low.fail		where Y = 1 or 2 for limit number
Remarks	This attribute returns the results of low limit Y testing: <ul style="list-style-type: none"> • 0 indicates test passed – measurement within the low limit • 1 indicates test failed – measurement has exceeded low limit A failed indication does indicate the low limit caused the failure. You may read the measurement event register of the status model to see fail indication as well. This response only has meaning if the limit's auto clear setting is disabled and the limit is enabled. Otherwise, it will indicate the results of the last limit test when enabled.	
Also see	digio.trigger[N].stimulus (on page 13-91)	
Example	To query the test results of low limit 2: <pre>results = dmm.limit[2].low.fail</pre>	

dmm.limit[Y].low.value		where Y = 1 or 2 for limit number
Attribute	Indicates the low limit value for limit Y.	
Usage	To read the low value of limit Y: <pre>value = dmm.limit[Y].low.value</pre> To write the low value of limit Y: <pre>dmm.limit[Y].low.value = value</pre> value: Valid range is -4294967295 to +4294967295	
Remarks	Use this attribute to specify or query the low limit value of limit Y. When limit Y testing is enabled (dmm.limit[Y].enable (on page 13-142)), the measurement value must be greater than this value to avoid causing a fail indication for the limit. The default value is -1 for limit 1 and -2 for limit 2. You may set or get the value regardless if the limit is set to a digio trigger stimulus (digio.trigger[N].stimulus = <code>dmm.trigger.EVENT_LIMIT1_LOW</code> or <code>dmm.trigger.EVENT_LIMIT2_LOW</code>).	
Also see	dmm.limit[Y].low.fail (on page 13-143) dmm.limit[Y].high.value (on page 13-142)	
Example	To set low limit of limit 2 to -5: <pre>dmm.limit[2].low.value = -5</pre>	

dmm.linesync	
Attribute	Attribute indicating whether line sync is used during the measurement.

dmm.linesync	
Usage	<p>To read the linesync state:</p> <pre>value = dmm.linesync</pre> <p>To write the linesync state:</p> <pre>dmm.linesync = value</pre> <p>value: Set to one of the following:</p> <ul style="list-style-type: none"> • dmm.ON or 1 to enable line sync • dmm.OFF or 0 to disable line sync
Remarks	<p>This attribute is only valid when <code>dmm.func = "dcvolts", "dcurrent", "twowireohms", "fourwireohms", "temperature", "continuity", and "commonsideohms."</code> All other functions generate an error and return nil when queried.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the line sync setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is <code>dmm.OFF</code>.</p> <p>Enabling this attribute will synchronize a trigger to the main power line crossings.</p>
Example	<p>To enable line sync on "fourwireohms"</p> <pre>dmm.func = "fourwireohms" dmm.linesync = dmm.ON</pre>

dmm.makebuffer()	
Function	Creates a user buffer for storing readings.
Usage	<pre>mybuffer = dmm.makebuffer(bufferize)</pre> <p>bufferize: Maximum number of readings that can be stored.</p>
Remarks	<p>These reading buffers are allocated dynamically. This function creates the buffers where <code>bufferize</code> indicates the maximum number of readings the buffer can store.</p> <p>These buffers can be deleted by setting <code>mybuffer</code> to nil.</p>

dmm.makebuffer()	
Details	<p>Once a buffer is created, the attributes are:</p> <ul style="list-style-type: none"> • <code>mybuffer.appendmode = 1 (ON) or 0 (OFF)</code> – default 0 over a bus interface, but 1 for ones created on the front panel • <code>mybuffer.basetimeseconds</code> returns the seconds for reading buffer entry 1 (read-only attribute). • <code>mybuffer.basetimefractional</code> returns the seconds and fractional seconds for reading buffer entry 1 (read-only attribute). • <code>mybuffer.capacity</code> for overall buffer size • <code>mybuffer.collecttimestamps = 1(ON) or 0(OFF)</code> – default 1 • <code>mybuffer.collectchannels = 1(ON) or 0(OFF)</code> – default 1 • <code>mybuffer.n</code> for number of readings stored in buffer currently • <code>mybuffer.timestampresolution</code> returns the resolution of the time stamping (read-only attribute). <p>The following buffer bits indicate buffer statuses:</p> <p><code>dmm.buffer.LIMIT1_LOW_BIT</code> or 1 <code>dmm.buffer.LIMIT1_HIGH_BIT</code> or 2 <code>dmm.buffer.LIMIT2_LOW_BIT</code> or 4 <code>dmm.buffer.LIMIT2_HIGH_BIT</code> or 8 <code>dmm.buffer.MEAS_OVERFLOW_BIT</code> or 64 <code>dmm.buffer.MEAS_CONNECT_QUESTION_BIT</code> or 128</p>
Details, continued	<p>To see readings in buffer:</p> <pre>printbuffer(x, y, mybuffer)</pre> <p>x and y: represent reading numbers desired</p> <p>To see readings, channels, and units:</p> <pre>printbuffer(x, y, mybuffer, mybuffer.channels, mybuffer.units)</pre> <p>x and y: represent reading numbers desired</p> <p>To see time stamps in buffer:</p> <pre>mybuffer.collecttimestamps = 1 print(x, y, mybuffer, mybuffer.timestamps)</pre> <p>x and y: represent readings and time stamps for elements x to y</p> <p>To see seconds, fractional seconds, and relative time stamps,</p> <pre>mybuffer.collecttimestamps = 1 printbuffer(x, y, mybuffer.seconds) printbuffer(x, y, mybuffer.fractionalseconds) printbuffer(x, y, mybuffer.relativetimestamps)</pre>

dmm.makebuffer()	
Also see	Reading buffers (on page 7-12) for more information on reading buffer aspects in the system
Example	To create a user reading buffer named <code>mybuffer2</code> , with a capacity of 300: <pre>mybuffer2 = dmm.makebuffer(300)</pre> To delete <code>mybuffer2</code> : <pre>mybuffer2 = nil</pre>

dmm.math.enable	
Attribute	Enable or disable math operation on measurements.
Usage	To read the math operation state: <pre>value = dmm.math.enable</pre> To write the math operation state: <pre>dmm.math.enable = value</pre> value: Set to one of the following: dmm.ON or 1 to enable math operation on measurements dmm.OFF or 0 to disable math operation on measurements
Remarks	When this attribute is set to dmm.ON, the math operation specified by math format attribute (dmm.math.format (on page 13-147)) will be performed before completing a measurement. To not perform a math operation on a measurement, set the attribute to dmm.OFF. The default setting is dmm.OFF.
Also see	dmm.math.format (on page 13-147)
Example	To enable math operation on measurements: <pre>dmm.math.enable = dmm.ON</pre>

dmm.math.format	
Attribute	Specifies the math operation to perform on measurements.
Usage	To read the math operation type: <pre>value = dmm.math.format</pre> To write the math operation type: <pre>dmm.math.format = value</pre> Set value to: <ul style="list-style-type: none"> • dmm.MATH_NONE or 0 • dmm.MATH_MXB or 1 • dmm.MATH_PERCENT or 2 • dmm.MATH_RECIPROCAL or 3

dmm.math.format	
Remarks	<p>To have no math operation performed on the measurements, set this attribute to <code>dmm.MATH_NONE</code>. Having this equal <code>dmm.MATH_NONE</code> and enabling math operation (<code>dmm.math.enable</code>), the equivalent effect is disabling math operation.</p> <p>Use a setting of <code>dmm.MATH_MXB</code> to have</p> $Y = mX + b$ <p>where X is the normal measurement</p> <p>m is user entered constant for scale factor (dmm.math.mxb.mfactor (on page 13-149))</p> <p>b is user entered constant for offset (dmm.math.mxb.bfactor (on page 13-148))</p> <p>Y is the result</p> <p>When using relative offset measurement control (dmm.rel.enable (on page 13-159)), the rel'ed reading is used for X.</p> <p>Use a setting of <code>dmm.MATH_PERCENT</code> to have:</p> $\text{Percent} = \frac{(\text{Input} - \text{Reference})}{\text{Reference}} \times 100\%$ <p>where:</p> <p>Input is the normal measurement (if using REL, it will be the Rel'ed value)</p> <p>Reference is user entered constant (dmm.math.percent (on page 13-150))</p> <p>Percent is the result</p> <p>Use a setting of <code>dmm.MATH_RECIPROCAL</code> for 1/X operation, where X is normal or REL'ed measurement value.</p> <p>The desired math operation is performed before any of the enabled limit testing. Default setting is <code>dmm.MATH_PERCENT</code>.</p>
Also see	<p>dmm.math.enable (on page 13-147)</p> <p>dmm.math.mxb.bfactor (on page 13-148)</p> <p>dmm.math.mxb.mfactor (on page 13-149)</p> <p>dmm.math.percent (on page 13-150)</p>
Example	<p>To enable the reciprocal operation on measurements:</p> <pre>dmm.math.format = dmm.MATH_RECIPROCAL dmm.math.enable = dmm.ON</pre>

dmm.math.mxb.bfactor	
Attribute	Specifies the offset for the $y = mx + b$ operation.
Usage	To read the offset for the $y = mx + b$ operation: <code>value = dmm.math.mxb.bfactor</code> To write the offset for the $y = mx + b$ operation: <code>dmm.math.mxb.bfactor = value</code> value: Valid range is -4294967295 to +4294967295
Remarks	This attribute specifies the offset (b) for an $mx + b$ operation.
Also see	dmm.math.format (on page 13-147) dmm.math.mxb.mfactor (on page 13-149)
Example	To set the offset for $mx + b$ operation to 50: <code>dmm.math.mxb.bfactor = 50</code>

dmm.math.mxb.mfactor	
Attribute	Specifies the scale factor for the $y = mx + b$ operation.
Usage	To read the scale factor for $y = mx + b$ operation: <code>value = dmm.math.mxb.mfactor</code> To write the scale factor for $y = mx + b$ operation: <code>dmm.math.mxb.mfactor = value</code> value: Valid range for value is -4294967295 to +4294967295
Remarks	This attribute represents the scale factor (m) for an $mx + b$ operation.
Details	dmm.math.format (on page 13-147) dmm.math.mxb.bfactor (on page 13-148)
Example	To set the scale factor for $mx + b$ operation to 0.80: <code>dmm.math.mxb.mfactor = 0.80</code>

dmm.math.mxb.units	
Attribute	Specifies the unit character for the $y = mx + b$ operation.
Usage	To read unit character for $y = mx + b$ operation: <code>value = dmm.math.mxb.units</code> To write unit character for $y = mx + b$ operation: <code>dmm.math.mxb.units = value</code> value: Valid characters, A to Z, '[' char for micro symbol (as in μV), use ']' char for ohm symbol (as in Ω) and use '\ ' char for degree symbol (as in $^\circ$).
Remarks	This attribute represents the unit character to use when the math format is set for $mx + b$ (<code>dmm.math.format = dmm.MATH_MXB</code>). Default setting is 'X'.

dmm.math.mxb.units	
Also see	dmm.math.format (on page 13-147)
Example	To set the units for mx +b operation to 'Q': <code>dmm.math.mxb.units = 'Q'</code>

dmm.math.percent	
Attribute	Specifies the constant to use for the percent operation.
Usage	To read the constant for the percent operation: <code>value = dmm.math.percent</code> To write the constant for the percent operation: <code>dmm.math.percent = value</code> value: Valid range is -4294967295 to +4294967295
Remarks	This attribute represents the constant to use for percent.
Also see	dmm.math.format (on page 13-147)
Example	To constant for percent operation to 1250: <code>dmm.math.percent = 1250</code> To acquire the percent constant: <code>dmm.math.percent = dmm.measure()</code>

dmm.measure()	
Function	Handles taking measurements on the DMM without using the trigger model.
Usage	There are two ways to use this function: To return the last reading of the measurement process taken and accounted for by dmm.measurecount: <code>reading = dmm.measure()</code> reading: Contains the last reading of the measurement process To return the last reading of the measurement process while storing all readings accounted for by dmm.measurecount: <code>reading = dmm.measure(rbuffer)</code> rbuffer: A previously created reading buffer object where all the reading(s) will be stored. reading: Contains the last reading of the measurement process

dmm.measure()	
Remarks	<p>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 the reading buffer if one is designated and has been created (see rbuffer in Usage), but only the last reading will be returned as reading.</p> <p>The dmm.measurecount (on page 13-151) attribute determines how many measurements are performed. When using a buffer, it also determines the number of readings to store in the buffer.</p>
Also see	<p>dmm.makebuffer() (on page 7-8)</p> <p>dmm.measurecount (on page 13-151)</p> <p>dmm.measurewithtime() (on page 13-151)</p>
Example	<p>To perform 100 DC voltage measurements and store them in a buffer called "mybuff":</p> <pre>mybuff = dmm.makebuffer(100) dmm.func = "dcvolts" dmm.measurecount = 100 dmm.measure(mybuff)</pre>

dmm.measurecount	
Attribute	Indicates the number of measurements to take when a measurement is requested by <code>dmm.measure</code> .
Usage	<p>To read measure count:</p> <pre>count = dmm.measurecount</pre> <p>To write measure count:</p> <pre>dmm.measurecount = count</pre> <p>count: Number of measurements. Maximum value: 450000</p>
Remarks	<p>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.</p> <p>It has no effect on the trigger model, and the trigger model does not affect this setting.</p> <p>The factory default and dmm.reset() (on page 13-161) value is 1.</p>
Also see	<p>dmm.makebuffer() (on page 7-8)</p> <p>dmm.measure() (on page 13-150)</p>
Example	<p>To set the measure count on DMM to 50:</p> <pre>dmm.measurecount = 50</pre>

dmm.measurewithtime()	
Function	Handles taking measurements on the DMM without using the trigger model and returns the reading along with time information.
Usage	<p>There are two ways to use this function:</p> <ul style="list-style-type: none"> To return the last reading of the measurement process taken and accounted for by <code>dmm.measurecount</code>: <code>reading, seconds, fractional = dmm.measurewithtime()</code> reading: Contains the last reading of the measurement process seconds: Contains seconds fractional: Contains fractional seconds To return the last reading of the measurement process while storing all readings accounted for by <code>dmm.measurecount</code>: <code>reading, seconds, fractional = dmm.measurewithtime(rbuffer)</code> reading: Contains the last reading of the measurement process seconds: Contains seconds fractional: Contains fractional seconds
Remarks	<p>This function returns only the last actual measurement and time information as <code>reading</code>, <code>seconds</code>, and <code>fractional seconds</code>. To use the additional information acquired while making a measurement, a reading buffer must be used. If the instrument is configured to return multiple readings when a measurement is requested, all readings will be available in <code>rbuffer</code> if one is provided, but only the last measurement and time information will be returned.</p> <p>The <code>dmm.measurecount</code> (on page 13-151) attribute determines how many measurements are performed. When using a buffer, it also determines the number of readings to store in the buffer.</p>
Also see	<p><code>dmm.makebuffer()</code> (on page 7-8)</p> <p><code>dmm.measure()</code> (on page 13-150)</p> <p><code>dmm.measurecount</code> (on page 13-151)</p>
Example	<p>To perform 100 DC voltage measurements and store them in a buffer called <code>mybuff</code>, and print the last measurement and time information:</p> <pre>mybuff = dmm.makebuffer(100) dmm.func = "dcvolts" dmm.measurecount = 100 reading, seconds, fractional = dmm.measurewithtime(mybuff) print(reading, seconds, fractional) ® -1.064005867e-002 1.779155900e+007 1.245658350e-001</pre>

dmm.nplc	
Attribute	Indicates the integration rate in line cycles for the DMM.
Usage	<p>To read the integration rate:</p> <pre>value = dmm.nplc</pre> <p>value: Represents the present integration rate in line cycles</p> <p>To write the integration rate:</p> <pre>dmm.nplc = value</pre> <p>value: Represents the desired integration in line cycles:</p> <ul style="list-style-type: none"> • 60 Hertz: 0.0005 to 15 • 50 Hertz: 0.0005 to 12
Remarks	<p>This is the integration rate setting for the DMM in line cycles. It applies to the selected function as indicated by dmm.func (on page 13-137). Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>The command generates an error when dmm.func is:</p> <ul style="list-style-type: none"> • "frequency" • "period" • "continuity" • "nofunction" <p>Also, an error will be generated if the value is out of range and dependent on the line frequency.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the integration rate for that function.</p> <p>The setting for NPLC may be adjusted based on what the DMM supports. Therefore, after setting the NPLC, query the value to see if it was adjusted.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 1.</p>
Also see	dmm.aperture (on page 13-110)
Example	<p>To set the NPLC for 2-wire ohms to 0.5:</p> <pre>dmm.func = "twowireohms"</pre> <pre>dmm.nplc = 0.5</pre>

dmm.offsetcompensation	
Attribute	Indicates the offset compensation setting for the DMM.
Usage	<p>To read the offset compensation:</p> <pre>value = dmm.offsetcompensation</pre> <p>value: Represents the present offset compensation setting.</p> <p>To write the offset compensation:</p> <pre>dmm.offsetcompensation = value</pre> <p>value: Represents the desired offset compensation setting. Set to one of the following:</p> <ul style="list-style-type: none"> • dmm.ON or 1 to enable offset compensation • dmm.OFF or 0 to disable offset compensation
Remarks	<p>This is the offset compensation setting for the DMM, and it applies to the selected function as indicated by dmm.func (on page 13-137). Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>The command applies when dmm.func is:</p> <ul style="list-style-type: none"> • "fourwireohms" • "commonsideohms" • "temperature" <p>When dmm.func = "temperature", the command applies only when the transducer type is 3 or 4-wire RTD (for all others, it is ignored). Set this command like you would for 4-wire ohm measurements.</p> <p>All other function settings will generate an error if the command is received. Also, an error will be generated if the value is invalid.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the offset compensation setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is dmm.OFF.</p>
Example	<p>To enable offset compensation for 4-wire ohms:</p> <pre>dmm.func = "fourwireohms" dmm.offsetcompensation = dmm.ON</pre>
dmm.open()	
Function	Opens the specified channel or channel pattern.
Usage	<pre>dmm.open (<ch_list>)</pre> <p>ch_list: string listing the channel or channel pattern to open.</p>

dmm.open()	
Remarks	<p>Aspects associated with this function include:</p> <ul style="list-style-type: none"> • Opening the channels and analog backplane relays for a measuring function. • The configuration (see dmm.getconfig() (on page 13-139)) associated with the specified channel dictates whether a paired channel is open or not. For channel patterns, the channels associated with it will be opened. Channel patterns don't support the concept of pairing channels for multi-wire measurements. • The configuration (see dmm.getconfig() (on page 13-139)) will dictate whether analog backplane relay 1 and 2 are opened. • Does not use analog backplane relays specified by channel.setbackplane() (on page 13-70) function or poles setting set by channel.setpole() (on page 13-79) function. <p>An error will be generated if:</p> <ul style="list-style-type: none"> • There is a syntax error in parameter string. • An empty parameter string is specified. • The specified channel or channel pattern is invalid. • A channel number does not exist for installed card in slot specified. • A slot is empty. • The channel pattern does not exist. • Does not support being closed like a digital I/O channel. • Channel is paired with another bank for a multi-wire application. • Channel's configuration is 'nofunction'. • More than one channel or channel pattern is specified in parameter. <p>Once an error is detected, the command stops processing. Channels open only if no error is detected. Otherwise, channels remain unchanged.</p> <p>This command allows you to separate the opening of channels with closing in a measuring aspect. Therefore, you may execute any number of commands between the open and close commands to satisfy your application needs.</p>
Also see	<p>channel.getclose() (on page 13-46)</p> <p>channel.getstate() (on page 13-56)</p> <p>dmm.close() (on page 13-123)</p>
Example	<p>To open Channel 3 on Slot 3:</p> <pre>dmm.open('3003')</pre> <p>To open a channel pattern called 'mychans':</p> <pre>dmm.open('mychans')</pre>

dmm.opendetector	
Attribute	Indicates the state of the temperature or 4-wire ohms open lead detector being used.
Usage	<p>To read the open lead detector state:</p> <pre>value = dmm.opendetector</pre> <p>value: Represents the present open detector state</p> <p>To write the open lead detector state:</p> <pre>dmm.opendetector = value</pre> <p>value: Represents the desired open detector state. Set to one of the following:</p> <ul style="list-style-type: none"> • dmm.ON or 1 to enable the open lead detector • dmm.OFF or 0 to disable the open lead detector
Remarks	<p>This attribute is valid when <code>dmm.func = "temperature"</code>. It also is valid when <code>dmm.func = "fourwireohms"</code>. All other configurations generate an error and return nil when queried (including "twowireohms"). When on temperature, the open detector setting is only used when the transducer type is thermocouple. For all other transducer types, it is ignored.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the open detector setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is <code>dmm.OFF</code>.</p>
Example	<p>To enable the thermocouple open detector:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THERMOCOUPLE dmm.opendetector = dmm.ON</pre>

dmm.range	
Attribute	Indicates the range of DMM for the selected function.
Usage	<p>To read the range for the selected function:</p> <pre>value = dmm.range</pre> <p>value: Represents the present range for the selected function.</p> <p>To write the range for the selected function:</p> <pre>dmm.range = value</pre> <p>value: Represents the expected measurement value or desired range for the selected function.</p>

dmm.range	
Remarks	<p>This attribute is the range setting for the selected function (see dmm.func (on page 13-137)) of the DMM. Querying the range when the selected function does not have a range associated with it will cause nil to be returned.</p> <p>The range will be selected based on which one best suits the expected measure value (value parameter). The command is applicable when <code>dmm.func = "dcvolts"</code> (0 to 303, default 303), <code>"acvolts"</code> (0 to 303, default 303), <code>"dcurrent"</code> (0 to 3.1, default 3.1), <code>"accurrent"</code> (0 to 3.1, default 3.1), <code>"twowireohms"</code> (0 to 120e6, default 120e6), <code>"fourwireohms"</code> (0 to 120e6, default 120e6) and <code>"commonsideohms"</code> (0 to 120e6, default 120e6).</p> <hr/> <p>NOTE The values in parentheses represent the valid range setting for each function.</p> <hr/> <p>An error will be generated if the command is received when dmm.func (on page 13-137) is:</p> <ul style="list-style-type: none"> • "temperature" • "frequency" • "period" • "continuity" • "nofunction". <p>Also, an error will be generated if parameter value does not make sense for selected function.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the range setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 303 because the default function is "dcvolts".</p> <p>Set this value to the expected measurement value and the unit will select the range appropriate to measure that value. Setting the range with this attribute will automatically disable the auto range setting (dmm.autorange (on page 13-113) command).</p>
Example	<p>To set the range for DC volts to 10:</p> <pre>dmm.func = "dcvolts" dmm.range = 5</pre>

dmm.refjunction	
Attribute	Indicates the type of the thermocouple reference junction.
Usage	<p>To read the reference junction type:</p> <pre>value = dmm.refjunction</pre> <p>value: Represents the present reference junction type.</p> <p>To write the reference junction type:</p> <pre>dmm.refjunction = value</pre> <p>value: Represents the desired reference junction type. Use one of the following values:</p> <ul style="list-style-type: none"> • dmm.REF_JUNCTION_SIMULATED or 0 • dmm.REF_JUNCTION_INTERNAL or 1 • dmm.REF_JUNCTION_EXTERNAL or 2
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. When on temperature, the setting is only applicable when the transducer type is set for thermocouple. For all other transducer types, the reference junction may be set, but will be ignored until the transducer type is set to thermocouple.</p> <p>Changing functions with <code>dmm.func</code> (on page 13-137) will reflect the reference junction setting for that function.</p> <p>The factory default and <code>dmm.reset()</code> (on page 13-161) function value is <code>dmm.REF_JUNCTION_INTERNAL</code>.</p>
Details	The default value is internal; change to simulated or external as needed.
Example	<p>To enable the simulated thermocouple reference junction:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THERMOCOUPLE dmm.refjunction = dmm.REF_JUNCTION_SIMULATED</pre>
dmm.rel.acquire()	
Function	Acquires an internal measurement to store as the relative (REL) level value.
Usage	<pre>rel_value = print(dmm.rel.acquire())</pre> <p>rel_value: The internal measurement acquired for the relative (REL) level value.</p>

dmm.rel.acquire()	
Remarks	<p>This function triggers the DMM to take a new measurement for the selected function. This measurement will then be stored as the new REL level setting.</p> <p>This function will return the acquired reading or nil, if an error occurred. An error will be generated if the active function does not support a REL level setting or the DMM is unable to take the measurement. When an error occurs, the REL level setting maintains the last valid setting.</p> <p>The command will generate an error when <i>dmm.func</i> (on page 13-137) equals one of the following:</p> <ul style="list-style-type: none"> • "continuity" • "nofunction" <p>After executing this command, use the <i>dmm.rel.level</i> (on page 13-160) attribute to see the last REL level value that was acquired or set by the user. Setting the REL level with this acquire function does not use the math, limit, and filter settings. It will be a calibrated reading as if these settings are disabled.</p>
Also see	dmm.rel.level (on page 13-160)
Example	<p>To acquire a REL level value for temperature:</p> <pre>dmm.func = "temperature" rel_value = dmm.rel.acquire()</pre>

dmm.rel.enable	
Attribute	Enables or disables relative measurement control for the DMM.
Usage	<p>To read the relative control state:</p> <pre>value = dmm.rel.enable</pre> <p>value: Represents the present relative enable setting.</p> <p>To write the relative control state:</p> <pre>dmm.rel.enable = value</pre> <p>value: Represents the desired relative measurement control setting. Set to one of the following:</p> <p>dmm.ON or 1 to enable relative measurements dmm.OFF or 0 to disable relative measurements</p>

dmm.rel.enable	
Remarks	<p>This is the relative measurement control setting for the DMM and it applies to the selected function as indicated by dmm.func (on page 13-137). Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>When relative measurements are enabled, all subsequent measured readings will be offset by the specified relative offset (see dmm.rel.level (on page 13-160)). Specifically, each returned measured relative reading will be the result of the following calculation:</p> <p>Relative reading = Actual measured reading – Relative offset value</p> <p>The command will generate an error when dmm.func equals one of the following:</p> <ul style="list-style-type: none"> • "continuity" • "nofunction" <p>Also, an error will be generated if the value is out of range for the selected function.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the relative enable setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is dmm.OFF.</p>
Also see	dmm.rel.level (on page 13-160)
Example	<p>To enable the relative measurements for AC current:</p> <pre>dmm.func = "accurrent" dmm.rel.enable = dmm.ON</pre>

dmm.rel.level	
Attribute	The offset value for relative measurements for the DMM.
Usage	<p>To read the relative offset level:</p> <pre>value = dmm.rel.level</pre> <p>value: Represents the present relative offset level.</p> <p>To write the relative offset level:</p> <pre>dmm.rel.level = value</pre> <p>value: Represents the desired relative offset level.</p>

dmm.rel.level	
Remarks	<p>This is the relative offset level setting for the DMM. It applies to the selected function as indicated by <i>dmm.func</i> (on page 13-137). Querying the setting when the selected function does not support it will cause nil to be returned.</p> <p>When relative measurements are enabled (see <i>dmm.rel.enable</i> (on page 13-159)), all subsequent measured readings will be offset by the specified relative offset value. Specifically, each returned measured relative reading will be the result of the following calculation:</p> <p>Relative reading = Actual measured reading – Relative offset value</p> <p>The command will generate an error when <i>dmm.func</i> (on page 13-137) = "continuity" or "nofunction". Also, an error will be generated if the value is out of range for the selected function.</p> <p>Changing functions with <i>dmm.func</i> (on page 13-137) will reflect the relative level offset setting for that function.</p> <p>The factory default and <i>dmm.reset()</i> (on page 13-161) function value is 0.</p> <hr/> <p>NOTE Using <code>dmm.rel.level = dmm.measure()</code> to set the REL level will include math, limits, and filter operations, if enabled. However, using the <i>dmm.rel.acquire()</i> (on page 13-158) function to set the REL level will not use these operations, even if enabled.</p>
Also see	<i>dmm.rel.enable</i> (on page 13-159)
Example	<p>To perform an AC current measurement and use it as the relative offset value:</p> <pre>dmm.func = "accurrent" dmm.rel.level = dmm.measure() -- see NOTE in Remarks</pre> <p>To acquire a relative offset value:</p> <pre>rel_value = dmm.measure() -- see NOTE in Remarks dmm.rel.level = rel_value</pre> <p>To acquire a fresh REL level after using <i>dmm.measure()</i> (on page 13-150) to set one:</p> <pre>dmm.rel.enable = dmm.OFF dmm.rel.level = dmm.measure() -- see NOTE in Remarks dmm.rel.enable = dmm.ON</pre> <p>To acquire a REL level value for temperature (using <i>dmm.rel.acquire()</i> (on page 13-158)):</p> <pre>dmm.func = "temperature" rel_value = dmm.rel.acquire()</pre>
dmm.reset()	
Function	Resets DMM aspects of the system, as indicated by the parameter.
Usage	<pre>dmm.reset(scope)</pre> <p>scope: A string equaling "active" for active function only to factory default settings or "all" for all functions back to factory default settings.</p>

dmm.reset()	
Remarks	<p>When the parameter equals "active", this command will only reset the DMM aspects of the system for the active function only. Settings affects are:</p> <ul style="list-style-type: none"> • Active dmm function (dmm.func (on page 13-137)) has its pertinent attributes reset to factory default values. • Other functions are unchanged. • DMM configurations (dmm.setconfig() (on page 13-168) and dmm.getconfig() (on page 13-139)) are unchanged. The rest of the settings are unaffected. To reset the entire system to factory default settings, use the reset() (on page 13-230) command. To reset all functions of the DMM, use the <code>dmm.reset("all")</code> command. <p>When the parameter equals "all", this command will only reset the dmm aspects of the system to factory default settings. Settings affected are</p> <ul style="list-style-type: none"> • Selected dmm function (dmm.func (on page 13-137)) goes to "dcvolts". • Dmm settings go to defaults for "dcvolts". • Each other function is reset back to its factory default settings as well. • The rest of the settings are unaffected. To reset the entire system to factory default settings, use the reset command. To reset the active function, use this command with the parameter set to "active".
Also see	reset() (on page 13-230)
Example	<p>To perform a reset on temperature only:</p> <pre>dmm.func = "temperature" dmm.reset("active")</pre> <p>To perform a reset on all DMM functions:</p> <pre>dmm.reset("all")</pre>

dmm.rtdalpha	
Attribute	Indicates the user type RTD alpha value.
Usage	<p>To read the user type RTD alpha value:</p> <pre>value = dmm.rtdalpha</pre> <p>value: Represents the present user alpha value.</p> <p>To write user type RTD alpha value:</p> <pre>dmm.rtdalpha = value</pre> <p>value: Represents the desired user alpha value.</p>

dmm.rtdalpha	
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. Errors are also generated if the parameter value is out of range. The valid range for user alpha is 0 to 0.01. For temperature, this setting is used when the transducer type is set to 3 or 4-wire RTD. For other transducer types, the setting will be set but ignored until the transducer type is set to an RTD type.</p> <hr/> <p>NOTE The following attributes share common settings and apply to both 3 and 4-wire RTDs: <code>dmm.rtdalpha</code>, dmm.rtdbeta (on page 13-163), dmm.rtddelta (on page 13-164), and dmm.rtdzero (on page 13-165). Therefore, when both 3 and 4-wire RTDs are set to USER type for RTD, switching transducers between 3 and 4 will cause both to use the same settings (for example, <code>dmm.rtdalpha</code>, <code>dmm.rtdbeta</code>). If unique settings are desired, they must be changed, or use two different DMM configurations.</p> <hr/> <p>Changing functions with dmm.func (on page 13-137) will reflect the RTD alpha setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 0.00385055.</p>
Also see	<p>dmm.rtdbeta (on page 13-163)</p> <p>dmm.rtddelta (on page 13-164)</p> <p>dmm.rtdzero (on page 13-165)</p>
Example	<p>To set user alpha constant for RTD to 0.005:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THREERTD or dmm.TEMP_FOURRTD dmm.rtdalpha = 0.005</pre>
dmm.rtdbeta	
Attribute	Indicates the user beta value for user type RTD.
Usage	<p>To read the user type RTD beta value:</p> <pre>value = dmm.rtdbeta</pre> <p>value: Represents the present user type RTD beta value.</p> <p>To write the user type RTD beta value:</p> <pre>dmm.rtdbeta = value</pre> <p>value: Represents the desired user type RTD beta value.</p>

dmm.rtdbeta	
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. Also, get errors if parameter value out of range. The valid range for user beta is 0 to 1.0. For temperature, this setting is used when the transducer type is set to 3 or 4-wire RTD. For other transducer types, the setting will be set but ignored until the transducer type is set to an RTD type.</p> <hr/> <p>NOTE The following attributes share common settings and apply to both 3 and 4-wire RTDs: dmm.rtdalpha (on page 13-162), <code>dmm.rtdbeta</code>, dmm.rtddelta (on page 13-164), and dmm.rtdzero (on page 13-165). Therefore, when both 3 and 4-wire RTDs are set to USER type for RTD, switching transducers between 3 and 4 will cause both to use the same settings (for example, <code>dmm.rtdalpha</code>, <code>dmm.rtdbeta</code>). If unique settings are desired, they must be changed, or use two different DMM configurations.</p> <hr/> <p>Changing functions with dmm.func (on page 13-137) will reflect the RTD beta setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 0.111.</p>
Also see	<p>dmm.rtdalpha (on page 13-162)</p> <p>dmm.rtddelta (on page 13-164)</p> <p>dmm.rtdzero (on page 13-165)</p>
Example	<p>To set user beta constant for RTD to 0.3:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THREERTD or dmm.TEMP_FOURRTD dmm.rtdbeta = 0.3</pre>

dmm.rtddelta	
Attribute	Indicates the user type RTD delta value.
Usage	<p>To read the user type RTD delta value:</p> <pre>value = dmm.rtddelta</pre> <p>value: Represents the present user type RTD delta value.</p> <p>To write the user type RTD delta value:</p> <pre>dmm.rtddelta = value</pre> <p>value: Represents the user type RTD delta value.</p>

dmm.rtddelta	
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. Also, get errors if parameter value out of range. The valid range for user delta is 0 to 5. For temperature, this setting is used when the transducer type is set to 3 or 4-wire RTD. For other transducer types, the setting will be set, but ignored until the transducer type is for an RTD one.</p> <hr/> <p>NOTE The following attributes share common settings and apply to both 3 and 4-wire RTDs: dmm.rtdalpha (on page 13-162), dmm.rtdbeta (on page 13-163), <code>dmm.rtddelta</code>, and dmm.rtdzero (on page 13-165). Therefore, when both 3 and 4-wire RTDs are set to USER type for RTD, switching transducers between 3 and 4 will cause both to use the same settings (for example, <code>dmm.rtdalpha</code>, <code>dmm.rtdbeta</code>). If unique settings are desired, they must be changed, or use two different DMM configurations.</p> <hr/> <p>Changing functions with dmm.func (on page 13-137) will reflect the RTD delta setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 1.507.</p>
Also see	<p>dmm.rtdalpha (on page 13-162)</p> <p>dmm.rtdbeta (on page 13-163)</p> <p>dmm.rtdzero (on page 13-165)</p>
Example	<p>To set user delta constant for RTD to 3:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THREERTD or dmm.TEMP_FOURRTD dmm.rtddelta = 3</pre>
dmm.rtdzero	
Attribute	Indicates the user type RTD zero value.
Usage	<p>To read the user type RTD zero value:</p> <pre>value = dmm.rtdzero</pre> <p>value: Represents the present user type RTD zero value.</p> <p>To write the user type RTD zero value:</p> <pre>dmm.rtdzero = value</pre> <p>value: Represents the desired user type RTD zero value.</p>

dmm.rtdzero	
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. Errors will also be generated if the parameter value is out of range. The valid range for user zero is 0 to 10000. For temperature, this setting is used when the transducer type is set to 3 or 4-wire RTD. For other transducer types, the setting will be set but ignored until the transducer type is set for an RTD type.</p> <hr/> <p>NOTE The following attributes share common settings and apply to both 3 and 4-wire RTDs: dmm.rtdalpha (on page 13-162), dmm.rtdbeta (on page 13-163), dmm.rtddelta (on page 13-164), and <code>dmm.rtdzero</code>. Therefore, when both 3 and 4-wire RTDs are set to USER type for RTD, switching transducers between 3 and 4 will cause both to use the same settings (for example, <code>dmm.rtdalpha</code>, <code>dmm.rtdbeta</code>). If unique settings are desired, they must be changed, or use two different DMM configurations.</p> <hr/> <p>Changing functions with dmm.func (on page 13-137) will reflect the RTD zero setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 100.</p>
Also see	<p>dmm.rtdalpha (on page 13-162)</p> <p>dmm.rtdbeta (on page 13-163)</p> <p>dmm.rtddelta (on page 13-164)</p>
Example	<p>To set user zero constant for RTD to 300:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THREERTD or dmm.TEMP_FOURRTD dmm.rtdzero = 300</pre>
dmm.savebuffer()	
Function	Saves data from the specified dynamically-allocated buffer to the USB flash drive using the specified filename.

dmm.savebuffer()	
Usage	<pre>dmm.savebuffer('<reading buffer name>', '<filename>', time_format)</pre> <p>reading buffer name: The name of a previously created DMM reading buffer, specified as a string. Do not pass the reading buffer name without quotes because this generates a data type error. For example, if the reading buffer is <code>mybuffer</code>, then the buffer name should be specified as <code>"mybuffer"</code> and not <code>mybuffer</code>.</p> <p>filename: The destination filename located on the USB flash drive. The filename must specify the full path (including <code>/usb1/</code>) and include the name of file with the file extension <code>.csv</code>. If no file extension is specified, <code>.csv</code> will be added to filename.</p> <p>time_format: This optional parameter indicates what date and time information should be saved in the file to the thumb drive. Use the following values for <code>time_format</code>:</p> <ul style="list-style-type: none"> • <code>dmm.buffer.SAVE_RELATIVE_TIME</code>, which saves relative time stamps only • <code>dmm.buffer.SAVE_FORMAT_TIME</code>, which is the default if no time format specified and saves dates, times and fractional seconds • <code>dmm.buffer.SAVE_RAW_TIME</code>, which saves seconds and fractional seconds only • <code>dmm.buffer.SAVE_TIMESTAMP_TIME</code>, which only saves time stamps <p>For options that save more than one item of time information, each item is comma delimited. For example, <code>dmm.buffer.SAVE_FORMAT_TIME</code> will have <code><date></code>, <code><time></code>, and <code><fractional seconds></code> for each reading.</p>
Remarks	<p>The first parameter (reading buffer name) represents the reading buffer to be saved. The second (filename) is the filename of file to save reading buffer data to on USB flash drive. The third parameter is optional and indicates how the date and time information from the buffer should be saved. For options that save more than one item of time information, each item is comma delimited. For example, the default format will have <code><date></code>, <code><time></code>, and <code><fractional seconds></code> for each reading.</p> <p>Errors will be generated if reading buffer does not exist or is not a DMM buffer, or if the destination filename is not specified correctly. The <code>.csv</code> is appended to the filename (unless the <code>.csv</code> is specified by user). Any specified file extension other than <code>.csv</code> will generate errors.</p> <p>Valid destination filename examples:</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata')</pre> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.csv')</pre> <p>Invalid destination filename examples:</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.')</pre> <p>-Invalid extension due to period by no following letters for extension.</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.txt')</pre> <p>-Invalid extension. Use <code>.csv</code> or do not specify (no period)</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.txt.csv')</pre> <p>-invalid extension because 2 periods specified (<code>mydata_txt.csv</code> would be correct).</p>

dmm.savebuffer()	
Example	<p>To save readings from valid DMM buffer named mybuffer with default time information to a file named <code>mydata.csv</code> on the USB flash drive:</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydata.csv')</pre> <p>To save readings from mybuffer with relative time stamps to a file named <code>mydatarel.csv</code> on the USB flash drive:</p> <pre>dmm.savebuffer('mybuffer', '/usb1/mydatarel.csv', dmm.buffer.SAVE_RELATIVE_TIME)</pre>
dmm.setconfig()	
Function	Sets the DMM configuration associated with items specified in parameter list.
Usage	<pre>dmm.setconfig(<ch_list>, dmm_config)</pre> <p>ch_list: A string listing the channels and channel patterns to modify.</p> <p>dmm_config: Name of DMM configuration to assign to items in <code>ch_list</code>.</p>

dmm.setconfig()	
Remarks	<p>This command will associate the specified DMM configuration (<code>dmm_config</code>) with the items specified in the parameter channel list (<code>ch_list</code>). The configuration being assigned determines whether analog backplane relay 1 or 2 get used, based on the function associated with the configuration when being assigned to a channel. For channel patterns, the pattern image must include the desired analog backplane relays along with the desired channels. This command has no effect on the poles setting for a channel (channel.setpole() (on page 13-79)) or analog backplane relays specified by channel.setbackplane() (on page 13-70) function.</p> <p>An error will be generated if:</p> <ul style="list-style-type: none"> • A syntax error exists in either parameter. • An empty parameter string for either parameter is specified. • There is more than one DMM configuration is specified. • A specified channel does not exist for the card installed on the slot specified. • An empty slot is specified. • The desired DMM functionality is not supported. • A specified channel is forbidden to close. • Is an analog backplane relay. • The specified channel pattern does not exist. • A non-existing DMM configuration was specified. • A matrix channel is in channel list parameter (for example, the Model 3730 is 6 x 16 high density matrix card, so an error will be generated if a Model 3730 channel is included in the channel list parameter). <p>Once an error is detected, the command stops processing and no channels or channel patterns are modified. The DMM configuration for channels or channel patterns update only if no syntax errors exist in parameter and all channels and channel patterns are valid for having desired DMM configuration.</p> <p>The factory-default setting is 'nofunction'. This setting must be changed to use a channel with the dmm.close() (on page 13-123) function. The <code>dmm.close</code> function generates an error if the associated function of the DMM configuration is 'nofunction'.</p>
Also see	<p>dmm.getconfig() (on page 13-139)</p> <p>channel.setpole() (on page 13-79)</p>
Example	<p>To assign 'mydcv' to Channels 1 to 100 on Slots 1 to 3:</p> <pre>dmm.setconfig('1001:3100', 'mydcv')</pre> <p>To assign factory default settings for 'dcvolts' to channels on Slot 5:</p> <pre>dmm.setconfig('slot5', 'dcvolts')</pre>

dmm.simreftemperature	
Attribute	Indicates the simulated reference temperature for thermocouples.
Usage	<p>To read the simulated reference temperature: <code>value = dmm.simreftemperature</code></p> <p>value: Represents the present simulated reference temperature</p> <p>To write the simulated reference temperature: <code>dmm.simreftemperature = value</code></p> <p>value: Represents the desired simulated reference temperature in Celsius (0°C to 65°C), in Fahrenheit (32°F to 149°F) or in Kelvin (273°K to 338°K).</p>
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. When the function is temperature, the simulated reference temperature is only used when the transducer type is thermocouples (see dmm.transducer (on page 13-173) attribute). For all other transducer types, the setting will be updated but ignored until the transducer type is set for thermocouples.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the simulated reference temperature setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 23°C.</p>
Example	<p>To set 30 degrees Celsius as the simulated reference temperature for thermocouples:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THERMOCOUPLE dmm.units = dmm.UNITS_CELSIUS dmm.simreftemperature = 30</pre>

dmm.thermistor									
Attribute	Indicates the type of thermistor to use.								
Usage	<p>To read the thermistor type: <code>value = dmm.thermistor</code></p> <p>value: Represents the present thermistor type in ohms. The value will be one of the following: 2200, 5000 or 10000.</p> <p>To write the thermistor type: <code>dmm.thermistor = value</code></p> <p>value: Represents the desired thermistor type in ohms (1950 to 10050). The value parameter is converted to 2200, 5000 or 10000 as follows:</p> <table border="1" data-bbox="535 1648 1482 1837"> <thead> <tr> <th>Parameter</th> <th>Converted value</th> </tr> </thead> <tbody> <tr> <td><3500</td> <td>2200</td> </tr> <tr> <td>Between 3500 and 7500</td> <td>5000</td> </tr> <tr> <td>≥ 7500</td> <td>10000</td> </tr> </tbody> </table>	Parameter	Converted value	<3500	2200	Between 3500 and 7500	5000	≥ 7500	10000
Parameter	Converted value								
<3500	2200								
Between 3500 and 7500	5000								
≥ 7500	10000								

dmm.thermistor	
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. When the function is temperature, the thermistor attribute is only used when the transducer type is set for thermistor. For all other transducer types, the setting will be updated but not used until thermistor is selected for the transducer type. See the attribute dmm.transducer (on page 13-173).</p> <p>Changing functions with dmm.func (on page 13-137) reflects the thermistor setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 5000 ohms.</p>
Example	<p>To set thermistor type to 3000:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THERMISTOR dmm.thermistor = 3000 -- this gets adjusted to 2200 print(dmm.thermistor) --> "2200"</pre>

dmm.thermocouple	
Attribute	Indicates the thermocouple type.
Usage	<p>To read the thermocouple type:</p> <pre>value = dmm.thermocouple</pre> <p>value: Represents the present thermocouple type</p> <p>To write the thermocouple type:</p> <pre>dmm.thermocouple = value</pre> <p>value: Represents the desired thermocouple type. For value, use one of the following:</p> <ul style="list-style-type: none"> • <code>dmm.THERMOCOUPLE_J</code> or 0 • <code>dmm.THERMOCOUPLE_K</code> or 1 • <code>dmm.THERMOCOUPLE_T</code> or 2 • <code>dmm.THERMOCOUPLE_E</code> or 3 • <code>dmm.THERMOCOUPLE_R</code> or 4 • <code>dmm.THERMOCOUPLE_S</code> or 5 • <code>dmm.THERMOCOUPLE_B</code> or 6 • <code>dmm.THERMOCOUPLE_N</code> or 7

dmm.thermocouple	
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. When the function is temperature, the thermocouple attribute is only used when the transducer type is thermocouples (see dmm.transducer (on page 13-173) attribute). For all other transducer types, the setting will be updated but ignored until the transducer type is set for thermocouples.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the thermocouple setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is <code>dmm.THERMOCOUPLE_K</code>.</p>
Example	<p>To set thermocouple type to J:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THERMOCOUPLE dmm.thermocouple = dmm.THERMOCOUPLE_J</pre>
dmm.threertd	
Attribute	Indicates the type of three-wire RTD being used.
Usage	<p>To read the 3-wire RTD type:</p> <pre>value = dmm.threertd</pre> <p>value: Represents the present type for 3-wire RTD.</p> <p>To write the 3-wire RTD type:</p> <pre>dmm.threertd = value</pre> <p>value: Represents the desired type for 3-wire RTD.</p> <p>Use one of the following for value:</p> <ul style="list-style-type: none"> • <code>dmm.RTD_PT100</code> or 0 for type PT100 • <code>dmm.RTD_D100</code> or 1 for type D100 • <code>dmm.RTD_F100</code> or 2 for type F100 • <code>dmm.RTD_PT385</code> or 3 for type PT385 • <code>dmm.RTD_PT3916</code> or 4 for type PT3916 • <code>dmm.RTD_USER</code> or 5 for user specified type
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried. When the function is temperature, the three-wire RTD is only used when the transducer type is three-wire RTD (see dmm.transducer (on page 13-173) attribute). For all other transducer types, the setting will be updated but ignored until the transducer type is set for three-wire RTD.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the three-wire RTD setting for that function.</p> <p>The dmm.reset() (on page 13-161) function will set this attribute to <code>dmm.RTD_PT100</code>.</p>

dmm.threertd	
Example	To set the type of three-wire RTD for PT3916: <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THREERTD dmm.threertd = dmm.RTD_PT3916</pre>
dmm.threshold	
Attribute	Indicates the threshold range.
Usage	To read the threshold setting: <pre>value = dmm.threshold</pre> <p>value: Represents the present threshold setting.</p> <p>To write the threshold setting: <pre>dmm.threshold = value</pre> <p>value: Represents the desired threshold setting. The range for continuity is from 1 to 1000Ω. For frequency and period, the range is from 0 to 303V.</p> </p>
Remarks	This attribute is only valid when <code>dmm.func</code> is equal to: <ul style="list-style-type: none"> • "frequency" • "period" • "continuity" <p>All other configurations generate an error and return nil when queried.</p> <p>For "frequency" and "period", this refers to a threshold voltage range (0 to 303, default 10). For "continuity", it refers to a threshold resistance in ohms (1 to 1000, default 10). Errors will be generated if the parameter value does not make sense for selected function.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the threshold setting for that function.</p> <p>The factory default and dmm.reset() (on page 13-161) function value is 10 for all functions that support this attribute.</p>
Example	To set the threshold range for frequency to 30: <pre>dmm.func = "frequency" dmm.threshold = 30</pre>

dmm.transducer	
Attribute	Indicates the transducer type.
Usage	<p>To read the transducer type: <code>value = dmm.transducer</code></p> <p>value: Represents the present transducer type.</p> <p>To write the transducer type: <code>dmm.transducer = value</code></p> <p>value: Represents the desired transducer type.</p> <p>For value, use one of the following:</p> <ul style="list-style-type: none"> • <code>dmm.TEMP_THERMOCOUPLE</code> or 1 • <code>dmm.TEMP_THERMISTOR</code> or 2 • <code>dmm.TEMP_THREERTD</code> or 3 • <code>dmm.TEMP_FOURRTD</code> or 4
Remarks	<p>This attribute is only valid when <code>dmm.func = "temperature"</code>. All other configurations generate an error and return nil when queried.</p> <hr/> <p>NOTE This attribute setting affects what other temperature-supported attributes get used. There are various attributes that are only applicable when the transducer type is a certain type. Although the transducer type needs to match for the attribute setting to be used, the transducer type does not need to match to change the setting. For example, the transducer type does not need to be set to <code>dmm.TEMP_FOURRTD</code> to change the <code>dmm.fourrtd</code> (on page 13-137) attribute setting.</p> <hr/> <p>Changing functions with <code>dmm.func</code> (on page 13-137) will reflect the transducer setting for that function.</p> <p>The factory default and <code>dmm.reset()</code> (on page 13-161) function value is <code>dmm.THERMOCOUPLE</code>.</p>
Example	<p>To set transducer to thermistor type:</p> <pre>dmm.func = "temperature" dmm.transducer = dmm.TEMP_THERMISTOR</pre>
dmm.units	
Attribute	Indicates the units for voltage and temperature measurements.

dmm.units	
Usage	<p>To read the units:</p> <pre>value = dmm.units</pre> <p>value: Represents the present units.</p> <p>To write the units:</p> <pre>dmm.units = value</pre> <p>value: Represents the desired units.</p> <p>For value, use one of the following:</p> <ul style="list-style-type: none"> • dmm.UNITS_VOLTS or 0 • dmm.UNITS_DECIBELS or 1 • dmm.UNITS_CELSIUS or 2 • dmm.UNITS_KELVIN or 3 • dmm.UNITS_FAHRENHEIT or 4
Remarks	<p>This attribute is only valid when <code>dmm.func</code> is equal to:</p> <ul style="list-style-type: none"> • "dcvolts" • "acvolts" • "temperature". <p>All other configurations generate an error and return nil when queried. Errors will also be generated if the parameter value does not make sense for selected function.</p> <p>The settings of <code>dmm.UNITS_VOLTS</code> and <code>dmm.UNITS_DECIBELS</code> apply when <code>dmm.func = "dcvolts"</code> or <code>"acvolts"</code>. Likewise, settings of <code>dmm.UNITS_FAHRENHEIT</code>, <code>dmm.UNITS_CELSIUS</code>, and <code>dmm.UNITS_KELVIN</code> apply when <code>dmm.func = "temperature"</code>.</p> <p>Changing functions with dmm.func (on page 13-137) will reflect the units setting for that function.</p> <p>The factory default and <code>dmm.reset</code> function value is <code>dmm.UNITS_VOLTS</code>.</p> <p>The factory default for "temperature" is <code>dmm.UNITS_CELSIUS</code>.</p>
Example	<p>To set units for temperature measurements to Fahrenheit (°F):</p> <pre>dmm.func = "temperature" dmm.units = dmm.UNITS_FAHRENHEIT</pre>

errorqueue functions and attributes

Use the functions and attributes in this group to read the entries in the error queue.

errorqueue.clear()	
Function	Clears all entries out of the error queue.
Usage	<code>errorqueue.clear()</code>
Remarks	This function removes all entries from the error queue.
Details	See Error and status messages (on page 17-1) and Status Model (on page 12-1).
Also see	errorqueue.count (on page 13-176), errorqueue.next() (on page 13-176)

errorqueue.count	
Attribute	The number of entries in the error queue.
Usage	<code>count = errorqueue.count</code>
Remarks	This attribute can be read to determine the number of messages in the error queue. This is a read-only attribute. Writing to this attribute will generate an error.
Details	See Error and status messages (on page 17-1) and Status Model (on page 12-1).
Also see	errorqueue.clear() (on page 13-176) errorqueue.next() (on page 13-176)
Example	<p>Reads number of entries in the error queue:</p> <pre>count = errorqueue.count print(count) → 4.000000e+00</pre> <p>The above output indicates that there are four entries in the event queue.</p>

errorqueue.next()	
Function	Reads an entry from the error queue.
Usage	<pre>errorcode, message, severity, node = errorqueue.next()</pre> <p>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.</p>

errorqueue.next()	
Remarks	<ul style="list-style-type: none"> • 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 order. • If there are no entries in the queue, code 0, "Queue Is Empty" is returned. • Returned severity levels include the following: <ul style="list-style-type: none"> 0 Informational: Indicates no error: "Queue is Empty". 10 Informational: Indicates a status or minor error. Examples: "Reading Available" and "Reading Overflow". 20 Recoverable: Indicates possible invalid user input. The 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". 40 Fatal: Indicates that the Series 3700 is non-operational and will require service. Contact information for service is provided at the front of this manual. <p>In an expanded system, each TSP-Link™ enabled instrument is assigned a node number. node returns the node number where the error originated.</p>
Details	See Error and status messages (on page 17-1) and Status Model (on page 12-1).
Also see	errorqueue.clear() (on page 13-176) errorqueue.count (on page 13-176)
Example	<p>Reads the oldest entry in the error queue:</p> <pre>errorcode, message = errorqueue.next() print(errorcode, message)</pre> <p>Output: 0.000000e+00 Queue Is Empty</p> <p>The above output indicates that the queue is empty.</p>

eventlog functions and attributes

Use the functions and attributes in this group to control (read, write, enable, count, etc.) the event log for LXI aspects.

LXI event log

The LXI event log of a Series 3700 monitors all LAN triggers that the instrument receives or generates. The LXI event log has nine comma-delimited fields. Below is an example entry to a LXI event log and a description of the log fields in order of appearance.

```
"17:26:35.690 10 Oct 2007, LAN0, 192.168.1.102, LXI, 0,
1192037132, 1192037155.733269000, 0, 0x0"
```

Field #	Field Value	Field Description
1	"17:26:35.690 10 Oct 2007"	Formatted UTC time in 24-hour format including fractional seconds.
2	"LAN0"	Event identifier. NOTE This event identifier is zero-based (LAN0-LAN7). When specifying the LAN trigger using <code>lan.trigger[N]</code> , the minimum value for N is 1. Therefore LAN0 to LAN 7 corresponds to <code>lan.trigger[1]</code> through <code>lan.trigger[8]</code> , respectively.
3	"192.168.1.102"	IP address of the device that issued the LAN trigger.
4	"LXI"	LXI version identifier. Currently only LXI is defined.
5	"0"	LXI Domain number.
6	"1192037132"	Sequence number provided by the device that issued the LAN trigger. This number is incremented after generation of each LAN trigger.
7	"1192037155.733269000"	PTP time formatted as a floating point number.
8	"0"	The "overflow" from PTP seconds. Currently, this is "0". Also referred to as IEEE-1588 Epoch.
9	"0x0"	Hex value of the flag field, which is the logical OR of several conditions (error=1, retransmission=2, hardware=4, acknowledgement=8).

eventlog.all()	
Function	Returns all messages and removes them from the event log.
Usage	<code>eventlog.all()</code>
Remarks	This function returns all of the messages (return order is from oldest to newest) from the event log and removes them from the log. The response is a string that has the messages delimited with a new line character.
Also see	LXI event log (on page 11-7) eventlog.enable (on page 13-179) eventlog.count (on page 13-179) eventlog.clear() (on page 13-178) eventlog.next() (on page 13-180)

eventlog.clear()	
Function	Clears the event log.
Usage	<code>eventlog.clear()</code>
Remarks	This attribute erases any messages contained in the event log.

eventlog.clear()	
Also see	LXI event log (on page 11-7) eventlog.enable (on page 13-179) eventlog.count (on page 13-179) eventlog.next() (on page 13-180) eventlog.all() (on page 13-178)

eventlog.count	
Attribute	Reads the number of events contained in the event log.
Usage	To read the number of events: <code>N = eventlog.count</code> N: The number of events contained in the event log.
Remarks	This attribute indicates the present number of events contained in the event log.
Also see	LXI event log (on page 11-7) eventlog.enable (on page 13-179) eventlog.clear() (on page 13-178) eventlog.next() (on page 13-180) eventlog.all() (on page 13-178)
Example	To display the present number of events contained the Series 3700 event log: <pre>print(eventlog.count)</pre>

eventlog.enable	
Attribute	Reads or controls event log status.
Usage	To read event log status: <code>status = eventlog.enable</code> To write event log status: <code>eventlog.enable = status</code> status: The enable status of the event log; Use one of the following: <ul style="list-style-type: none"> • <code>eventlog.ENABLE</code> or 1: event log enable • <code>eventlog.DISABLE</code> or 0: event log disable
Remarks	This attribute indicates or controls the present status of the Series 3700 event log. When the event log is disabled (<code>eventlog.DISABLE</code> or 0), no events will be added to the event log.

eventlog.enable	
Also see	LXI event log (on page 11-7) eventlog.count (on page 13-179) eventlog.clear() (on page 13-178) eventlog.next() (on page 13-180) eventlog.all() (on page 13-178)
Example	To display the present status of the Series 3700 event log: <pre>print(eventlog.enable)</pre>

eventlog.next()	
Function	Returns the oldest message from the event log and removes it.
Usage	<code>eventlog.next()</code>
Remarks	This function returns the oldest message from the event log and removes it. The message is returned as a string.
Also see	LXI event log (on page 11-7) eventlog.enable (on page 13-179) eventlog.count (on page 13-179) eventlog.clear() (on page 13-178) eventlog.all() (on page 13-178)

exit functions

Use this function to terminate a script that is presently running.

exit()	
Function	Stops execution of a script.
Usage	<code>exit()</code>
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() (on page 3-16) function prior to calling exit.

file functions

Use the file commands when you need to manipulate file input or output with Series 3700 instruments. These commands reside in the file descriptors and operate exclusively on the file with which they are associated.

file:close()	
Function	Closes a file after flushing any data that was written to it with <code>io.write()</code> (on page 9-14) or <code>file:write()</code> (on page 9-11).
Usage	<code>file:close()</code> file: The descriptor of the file to close.
Remarks	This command is equivalent to <code>io.close(file)</code> . It is not remotely accessible.

file:flush()	
Function	Flush the buffered data for the specified file
Usage	<code>file:flush()</code> file: The descriptor of the file to flush
Remarks	Use this command to flush data written to it by <code>file:write()</code> (on page 9-11) or <code>io.write()</code> (on page 9-14). Using this function removes the need to close a file after writing to it and allows it to be left open to write more data. Data may be lost if the file is not closed or flushed before an application ends. To prevent the loss of data if there is going to be a time delay before more data is written when you want to keep file open and not close it, flush the file after writing to it.

file:read()	
Function	Reads data from a file.
Usage	<code>data = file:read(format)</code> data: The data read from the file. The number of return values matches the number of values in <code>format</code> . file: The descriptor of the file to read. format: A string or number indicating the type of data to be read. Any number of format parameters may be passed to this command, each corresponding to a returned data value. The format attribute is optional; the default is "*".

file:read()	
Remarks	<p>The <code>format</code> parameters may be any of the following:</p> <p>"*n": Return a number.</p> <p>"*a": Return the whole file, starting at the current position; return the empty string at the end of the file.</p> <p>"*l": Return the next line, skipping the end of line; return <code>nil</code> at the end of file.</p> <p>n: Return a string with up to <code>n</code> characters; return an empty string if <code>n</code> is zero; return <code>nil</code> at the end of file.</p> <p>Any error encountered is logged to the error queue.</p> <p>This command is not remotely accessible.</p>
file:seek()	
Function	Sets and gets a file's current position.
Usage	<p><code>position = file:seek(whence, offset)</code></p> <p>position: The new file position, measured in bytes from the beginning of the file.</p> <p>file: The descriptor of the file.</p> <p>whence: A string indicating the base against which offset is applied. The <code>whence</code> attribute is optional; the default is "cur".</p> <p>offset: The intended new position, measured in bytes from a base indicated by <code>whence</code>. Optional, default is 0.</p>
Remarks	<p>The <code>whence</code> parameters may be any of the following:</p> <p>"set": Beginning of file.</p> <p>"cur": Current position.</p> <p>"end": End of file.</p> <p>If an error is encountered, it is logged to the error queue, and the command returns <code>nil</code> and the error string.</p> <p>This command is not remotely accessible.</p>
file:write()	
Function	<p>Buffer data until a flush (<code>file:flush()</code> (on page 9-10) or <code>io.flush()</code> (on page 9-12)) or close (<code>file:close()</code> (on page 9-10) or <code>io.close()</code> (on page 9-12)) operation is performed.</p> <hr/> <p>NOTE Data may be lost if the file is not flushed or closed before the application ends. A write function buffers the data until a flush or close operation is requested.</p> <hr/>
Usage	<p><code>file:write(data)</code></p> <p>file: The descriptor of the file.</p> <p>data: The data to write to the file. An arbitrary number of data values may be passed to this command. All parameters must be either strings or numbers.</p>

file:write()	
Remarks	Any error encountered is logged to the error queue. This command is not remotely accessible.

format attributes

Use the format attributes to configure the output formats used by the [printnumber\(\)](#) (on page 13-222) and [printbuffer\(\)](#) (on page 13-221) functions. These attributes can 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	To read precision: <code>precision = format.asciiprecision</code> To write precision: <code>format.asciiprecision = precision</code> precision: Set from 1 to 16. Default value: 10.
Remarks	<ul style="list-style-type: none"> This attribute selects the precision (number of digits) for data printed with the printnumber() (on page 13-222), and printbuffer() (on page 13-221) 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 and reset precision is 10. Overflow readings (9.9E+37) may not appear as expected when ASCII precision is set to 1 or 16 (the extreme values).
Also see	format.byteorder (on page 13-183) format.data (on page 13-184) printbuffer() (on page 13-221) printnumber() (on page 13-222)
Example	Sets the ASCII precision to 7 digits and prints a number: <pre>format.asciiprecision = 7 print(2.5)</pre> Output: 2.500000E+00

format.byteorder	
Attribute	The binary byte order for data printed using the <code>printnumber</code> and <code>printbuffer</code> functions.
Usage	<p>To read byte order:</p> <pre>order = format.byteorder</pre> <p>To write byte order:</p> <pre>format.byteorder = order</pre> <p>Set order to one of the following values:</p> <ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • This attribute selects the byte order that data is written when printing data values with the <code>printnumber()</code> (on page 13-222) and <code>printbuffer()</code> (on page 13-221) 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 a PC-compatible computer.
Also see	<p>format.asciiprecision (on page 13-183)</p> <p>format.data (on page 13-184)</p> <p>printbuffer() (on page 13-221)</p> <p>printnumber() (on page 13-222)</p>
Example	<p>Selects the SWAPPED byte order:</p> <pre>format.byteorder = format.SWAPPED</pre>
format.data	
Attribute	The data format for data printed using the <code>printnumber</code> and <code>printbuffer</code> functions.

format.data	
Usage	<p>To read data format: <code>fmt = format.data</code></p> <p>To write data format: <code>format.data = fmt</code></p> <p>fmt: Set to one of the following values:</p> <p>1 or format.ASCII ASCII format.</p> <p>2 or format.SREAL Single precision IEEE-754 binary format.</p> <p>2 or format.REAL32 Single precision IEEE-754 binary format.</p> <p>3 or format.REAL Double precision IEEE-754 binary format.</p> <p>3 or format.REAL64 Double precision IEEE-754 binary format.</p>
Remarks	<ul style="list-style-type: none"> • This attribute selects the data format used to print data values with the printnumber() (on page 13-222) and printbuffer() (on page 13-221) functions. • The precision of the ASCII format can be controlled with the format.asciiprecision (on page 13-183) attribute. The byte order of SREAL, REAL, REAL32, and REAL64 can be selected with the format.byteorder (on page 13-183) 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	<p>format.asciiprecision (on page 13-183)</p> <p>format.byteorder (on page 13-183)</p> <p>printbuffer() (on page 13-221)</p> <p>printnumber() (on page 13-222)</p>
Example	<p>Selects the ASCII data format:</p> <pre>format.data = format.ASCII</pre>

fs functions

Use the fs commands to navigate the file system and list the available files on a flash drive. These commands are part of the Lua FS library.

fs.chdir()	
Function	Sets the current working directory.
Usage	<code>fs.chdir(path)</code> path : The new working directory path (absolute or relative).
Remarks	An error is logged to the error queue if the given path does not exist.
fs.cwd()	
Function	Returns the absolute path of the current working directory.
Usage	<code>path = fs.cwd()</code> path : The absolute path of the current working directory.
fs.is_dir()	
Function	Tests whether the specified path refers to a directory.
Usage	<code>status = fs.is_dir(path)</code> status : True if the given path is a directory; otherwise, false. path : The file system entry path (absolute or relative) to test.
Remarks	An error is logged to the error queue if the given path does not exist.
fs.is_file()	
Function	Tests whether the specified path refers to a file (as opposed to a directory).
Usage	<code>status = fs.is_file(path)</code> status : True if the given path is a file; otherwise, false. path : The path of the file system entry to test. This path may be absolute or relative to the current working directory.
Remarks	An error is logged to the error queue if the given path does not exist.
fs.mkdir()	
Function	Creates a directory at the specified path.
Usage	<code>fs.mkdir(path)</code> path : The path of the new directory. This path may be absolute or relative to the current working directory.
Remarks	An error is logged to the error queue if the parent folder of the new directory does not exist, or if a file system entry already exists at the given path.

fs.readdir()	
Function	Returns a list of all the file system entries within a specified directory.
Usage	<pre>files = fs.readdir(path)</pre> <p>files: A list containing the names of all the file system entries that reside in the specified directory.</p> <p>path: The directory path. This path may be absolute or relative to the current working directory.</p>
Remarks	This command is non-recursive (that is, entries in subfolders are not returned). An error is logged to the error queue if the given path does not exist, or does not represent a directory.

fs.rmdir()	
Function	Removes a directory from the file system.
Usage	<pre>fs.rmdir(path)</pre> <p>path: The path of the directory to remove. This path may be absolute or relative to the current working directory.</p>
Remarks	An error is logged to the error queue if the given path does not exist, does not represent a directory, or if the directory is not empty.

gpib attributes

Use the following attribute to set the GPIB address.

gpib.address	
Attribute	GPIB address.
Usage	<p>To read the GPIB address:</p> <pre>address = gpib.address</pre> <p>To write the GPIB address:</p> <pre>gpib.address = address</pre> <p>address: Set from 0 to 30. Default is 16.</p>
Remarks	<ul style="list-style-type: none"> • 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.

gpib.address	
Example	To set the GPIB address of the Series 3700 to 26 and then read the address: <pre>gpib.address = 26 address = gpib.address print(address) → 2.600000e+01</pre>

io functions

Use the io commands when you need to manipulate file input or output with Series 3700 instruments. These commands open and close file descriptors and perform basic I/O operations on a pair of default files, one input and one output.

io.close()	
Function	Closes the specified file.
Usage	<code>io.close(file)</code> file: A file descriptor to flush and close
Remarks	This command is equivalent to file:close() (on page 9-10).

io.flush()	
Function	Flush the buffered data for the current output file.
Usage	<code>io.flush()</code>
Remarks	Use this command to flush data written to the current default file by <code>file:write()</code> (on page 9-11) or <code>io.write()</code> (on page 9-14). Using this command removes the need to close a file after writing to it and allows it to be left open to write more data. Data may be lost if the file is not closed or flushed before an application ends. To prevent the loss of data if there is going to be a time delay before more data is written when you want to keep file open and not close it, flush the file after writing to it.

io.input()	
Function	Assigns a previously opened file, or opens a new file, as the default input file.
Usage	<code>io.input(filein)</code> <code>fileout = io.input()</code> filein: A file descriptor to assign or the path of a file to open as the default input file. The path may be absolute or relative to the current working directory. This parameter is optional; if absent, the command returns the absolute path to the current default input file (fileout). fileout: The absolute path to the default input file.
Remarks	Any error encountered is logged to the error queue. The remotely-accessible version of this command does not accept a file descriptor parameter.

io.open()	
Function	Opens a file for later access.
Usage	<pre>file, err, errnum = io.open(path, mode)</pre> <p>file: The descriptor of the opened file.</p> <p>err: A string with an error message an error occurred.</p> <p>errnum: Number representing the error number.</p> <p>path: The path of the file to open. This path may be absolute or relative to the current working directory.</p> <p>mode: A string representing the intended access mode. The mode attribute is optional; the default is "r".</p>
Remarks	<p>The <code>mode</code> string can be any of the standard C language <code>fopen</code> modes, including:</p> <p>"r": Read mode.</p> <p>"w": Write mode.</p> <p>"a": Append mode.</p> <p>If an error is encountered, it is logged to the error queue, and the command returns <code>nil</code> and the error string.</p> <p>This command is not remotely accessible.</p>

io.output()	
Function	Assigns a previously opened file or opens a new file as the default output file.
Usage	<pre>io.output(filein)</pre> <pre>fileout = io.output()</pre> <p>filein: A file descriptor to assign, or the path of a file to open, as the default output file. The path may be absolute or relative to the current working directory. This parameter is optional; if absent, the command returns the absolute path to the current default output file (<code>fileout</code>).</p> <p>fileout: The absolute path to the default output file.</p>
Remarks	<p>Any error encountered is logged to the error queue.</p> <p>The remotely-accessible version of this command does not accept a file descriptor parameter.</p>

io.read()	
Function	Reads data from the default input file.
Usage	<pre>data = io.read(format)</pre> <p>data: The data read from the file. The number of return values matches the number of values in <code>format</code>.</p> <p>format: A string or number indicating the type of data to be read. Any number of format parameters may be passed to this command, each corresponding to a returned data value. Optional; default is "*"l".</p>

io.read()	
Remarks	<p>The <code>format</code> parameters may be any of the following:</p> <p>"*n": Return a number.</p> <p>"*a": Return the whole file, starting at the current position; return an empty string at the end of file.</p> <p>"*l": Return the next line, skipping the end of line; return <code>nil</code> at the end of file.</p> <p>n: Return a string with up to <code>n</code> characters; return an empty string if <code>n</code> is zero; return <code>nil</code> at the end of file.</p> <p>Any error encountered is logged to the error queue.</p>

io.type()	
Function	Checks whether <code>obj</code> is a valid file handle.
Usage	<code>io.type(obj)</code>
Remarks	Returns "file" if <code>obj</code> is an open file handle, "closed file" if <code>obj</code> is a closed file handle, and <code>nil</code> if <code>obj</code> is not a file handle.

io.write()	
Function	<p>Buffer data until a flush (<code>file:flush()</code> (on page 9-10) or <code>io.flush()</code> (on page 9-12)) or close (<code>file:close()</code> (on page 9-10) or <code>io.close()</code> (on page 9-12)) operation is performed.</p> <hr/> <p>NOTE Data may be lost if the file is not flushed or closed before the application ends. A write buffers the data until a flush or close operation is requested.</p>
Usage	<p><code>io.write(data)</code></p> <p>data: The data to write to the file. An arbitrary number of data values may be passed to this command. All parameters must be either strings or numbers.</p>
Remarks	Any error encountered is logged to the error queue.

LAN functions and attributes

Use the functions and attributes in this group to set/read the LAN triggers, as well as control how LAN aspects of instrument are controlled. Use the `lan.config` functions and attributes to make changes before having them take effect. Use the `lan.status` functions and attributes to query for current settings.

lan.applysettings()	
Function	Reinitializes the LAN interface with new settings.
Usage	<code>lan.applysettings()</code>

lan.applysettings()	
Remarks	<p>This function disconnects from the LAN interface and reinitializes the LAN with the current configuration settings. This function initiates an overlapped operation. LAN configuration could be a lengthy operation. Although the function returns immediately, the LAN initialization will continue to run in the background.</p> <hr/> <p>NOTE When this command is executed, all existing LAN connections to the instrument will be disconnected.</p> <p>NOTE Even though the LAN configuration settings may not have changed since the LAN was last connected, new settings may take effect due to the dynamic nature of DHCP or DLLA configuration.</p> <hr/>
Example	<p>To reinitialize the LAN interface with new settings:</p> <pre>lan.applysettings()</pre>

lan.config.autonegotiate	
Attribute	Configures LAN auto-negotiation state.
Usage	<p>To read the auto-negotiation state:</p> <pre>state = lan.config.autonegotiate</pre> <p>To write the auto-negotiation state:</p> <pre>lan.config.autonegotiate = state</pre> <p>state: LAN auto-negotiation state. state may be one of the following values:</p> <ul style="list-style-type: none"> lan.ENABLE or 1: Enables auto-negotiation. lan.DISABLE or 0: Disables auto-negotiation.
Remarks	This attribute sets the LAN auto-negotiation state. When enabled, the unit will select the best options for Ethernet speed and duplex. By default, this feature is enabled.
Details	Changing this setting from lan.DISABLE to lan.ENABLE will cause the unit to immediately negotiate new speed and duplex settings. Changing this setting from lan.ENABLE to lan.DISABLE will cause the unit to immediately use the manually configured speed and duplex settings.
Example	<p>To enable and view LAN auto-negotiation:</p> <pre>lan.config.autonegotiate = lan.ENABLE print(lan.config.autonegotiate) → 1.000000000e+000</pre>

lan.config.dns.address[index]	
Attribute	Configures DNS server IP addresses.

lan.config.dns.address[index]	
Usage	<p>To read the IP address: <code>dnsaddress = lan.config.dns.address[index]</code></p> <p>To write the IP address: <code>lan.config.dns.address[index] = dnsaddress</code></p> <p>index: Entry index (1 or 2) dnsaddress: DNS server IP address.</p>
Remarks	<p>This attribute is an array of DNS server addresses. These addresses take priority for DNS lookups and will be consulted before any server addresses obtained using DHCP. This allows local DNS servers to be specified that take priority over DHCP configured global DNS servers.</p> <p>Up to two addresses may be specified. The address specified by index 1 will be consulted first for DNS lookups (the [index] must be either 1 or 2).</p>
Details	<p>NOTE Unused entries will be returned as "0.0.0.0" when read.dnsaddress must be a string specifying the DNS server's IP address in dotted decimal notation. To disable an entry, set its value to "0.0.0.0" or the empty string "".</p> <p>NOTE Although only two address may be manually specified here, the unit will use up to three DNS server addresses. If two are specified here, only one given by a DHCP server will be used. If no entries are specified here, up to three address given by a DHCP server will be used.</p>
Also see	<p>lan.config.dns.domain (on page 13-192) lan.config.dns.dynamic (on page 13-193) lan.config.dns.hostname (on page 13-193) lan.config.dns.verify (on page 13-194)</p>
Example	<p>To write a DNS address of 164.109.48.173 as address 1: <code>dnsaddress = '164.109.48.173'</code> <code>lan.config.dns.address[1] = dnsaddress</code></p>

lan.config.dns.domain	
Attribute	Configures Dynamic DNS domain.
Usage	<p>To read the present dynamic DNS domain: <code>domain = lan.config.dns.domain</code></p> <p>To write the dynamic DNS domain: <code>lan.config.dns.domain = domain</code></p> <p>domain: Dynamic DNS registration domain. Use a string of 255 characters or less.</p>

lan.config.dns.domain	
Remarks	<p>This attribute holds the domain to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the domain specified in this attribute with the DNS server.</p> <hr/> <p>NOTE The length of the fully qualified host name (combined length of the domain and hostname with separator character) must be less than or equal to 255 characters. Although up to 255 characters can be given here, care must be taken to be sure the combined length is also no more than 255 characters.</p> <hr/>
Also see	<p>lan.config.dns.dynamic (on page 13-193)</p> <p>lan.config.dns.hostname (on page 13-193)</p> <p>lan.config.dns.verify (on page 13-194)</p>
Example	<p>To display the present dynamic DNS domain:</p> <pre>print(lan.config.dns.domain)</pre>
lan.config.dns.dynamic	
Attribute	Configures Dynamic DNS registration state.
Usage	<p>To read Dynamic DNS registration state:</p> <pre>state = lan.config.dns.dynamic</pre> <p>To write Dynamic DNS registration state:</p> <pre>lan.config.dns.dynamic = state</pre> <p>state: Represents Dynamic DNS registration state. It may be one of the following values:</p> <ul style="list-style-type: none"> lan.ENABLE or 1: Enables dynamic DNS registration. lan.DISABLE or 0: Disables dynamic DNS registration.
Remarks	This attribute is used to enable or disable dynamic DNS registration. Dynamic DNS registration works with DHCP to register the hostname specified in the lan.config.dns.hostname attribute with the DNS server.
Also see	lan.config.dns.hostname (on page 13-193)
Example	<p>To display the Dynamic DNS registration state:</p> <pre>print(lan.config.dns.dynamic)</pre>

lan.config.dns.hostname	
Attribute	Configures Dynamic DNS hostname.
Usage	<p>To read Dynamic DNS hostname:</p> <pre>hostname = lan.config.dns.hostname</pre> <p>To write Dynamic DNS hostname:</p> <pre>lan.config.dns.hostname = hostname</pre> <p>hostname: Hostname to use for dynamic DNS registration. The hostname:</p> <ul style="list-style-type: none"> • must be a string of 255 characters or less • start with a letter • end with a letter or digit • contain only letters, digits, and hyphens
Remarks	<p>This attribute holds the hostname to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the hostname specified in this attribute with the DNS server.</p> <hr/> <p>NOTE The length of the fully qualified host name (combined length of the domain and hostname with separator character) must be less than or equal to 255 characters. Although up to 255 characters can be given here, care must be taken to be sure the combined length is also no more than 255 characters.</p> <hr/>
Also see	lan.config.dns.dynamic (on page 13-193)
Example	<p>To display the present Dynamic DNS hostname:</p> <pre>print(lan.config.dns.hostname)</pre>

lan.config.dns.verify	
Attribute	Configures DNS hostname verification state.
Usage	<p>To read DNS hostname verification state:</p> <pre>state = lan.config.dns.verify</pre> <p>To write DNS hostname verification state:</p> <pre>lan.config.dns.verify = state</pre> <p>state: DNS hostname verification state.state may be one of the following values:</p> <ul style="list-style-type: none"> • lan.ENABLE or 1: Enables DNS hostname verification. • lan.DISABLE or 0: Disables DNS hostname verification.
Remarks	<p>This attribute is used to enable or disable DNS hostname verification. If enabled, the unit will perform DNS lookups to verify that the DNS hostname matches the value specified by lan.config.dns.hostname (on page 13-193).</p>
Also see	lan.config.dns.hostname (on page 13-193)
Example	<p>To display present DNS hostname verification state:</p> <pre>print(lan.config.dns.verify)</pre>

lan.config.duplex	
Attribute	Configures LAN duplex mode.
Usage	<p>To read LAN duplex mode:</p> <pre>duplex = lan.config.duplex</pre> <p>To write LAN duplex mode:</p> <pre>lan.config.duplex = duplex</pre> <p>duplex: LAN duplex setting can be one of the following values:</p> <ul style="list-style-type: none"> lan.FULL or 1: Selects full-duplex operation. lan.HALF or 0: Selects half-duplex operation.
Remarks	<p>This attribute selects which duplex mode will be used by the LAN interface when lan.config.autonegotiate (on page 13-191) is disabled. When lan.config.autonegotiate (on page 13-191) is enabled, this setting is ignored.</p> <hr/> <p>NOTE This attribute does not indicate the actual setting currently in effect. Use the lan.status attributes to determine the current operating state of the LAN.</p>
Also see	lan.status
Example	<p>To display present LAN duplex mode:</p> <pre>print(lan.config.duplex)</pre>
lan.config.gateway	
Attribute	Configures LAN default gateway address.
Usage	<p>To read LAN gateway address:</p> <pre>gatewayaddress = lan.config.gateway</pre> <p>To write LAN gateway address:</p> <pre>lan.config.gateway = gatewayaddress</pre> <p>gatewayaddress: LAN default gateway address.</p>
Remarks	<p>This attribute specifies the default gateway IP address to use when manual or DLLA configuration methods are used to configure the LAN. This setting is ignored when DHCP is used.</p> <hr/> <p>NOTE This attribute does not indicate the actual setting currently in effect. Use the lan.status attributes to determine the current operating state of the LAN. gatewayaddress must be a string specifying the default gateway's IP address in dotted decimal notation.</p>
Example	<p>To display present gateway address:</p> <pre>print(lan.config.gateway)</pre>

lan.config.ipaddress	
Attribute	Configures LAN IP address.
Usage	To read the LAN IP address: <pre>ipaddress = lan.config.ipaddress</pre> To write the LAN IP address: <pre>lan.config.ipaddress = ipaddress</pre> ipaddress: LAN IP address.
Remarks	This attribute specifies the LAN IP address to use when the manual configuration method is used to configure the LAN. This setting is ignored when DLLA or DHCP is used. NOTE This attribute does not indicate the actual setting currently in effect. Use the lan.status attributes to determine the current operating state of the LAN. ipaddress must be a string specifying the IP address in dotted decimal notation.
Also see	lan.status.ipaddress (on page 13-201)
Example	To read the presently set LAN IP address: <pre>ipaddress = lan.config.ipaddress</pre>

lan.config.method	
Attribute	Controls LAN settings configuration method.
Usage	To read the method used: <pre>method = lan.config.method</pre> To write the method: <pre>lan.config.method = method</pre> method: LAN settings configuration method. It can be one of the following values: <ul style="list-style-type: none"> lan.AUTO or 1: Selects automatic sequencing of configuration methods. lan.MANUAL or 0: Use only manually specified configuration settings.
Remarks	This attribute controls how the LAN IP address, subnet mask, default gateway address, and DNS server addresses are determined. When method is lan.AUTO, the instrument will first attempt to configure the LAN settings using Dynamic Host Configuration Protocol (DHCP). If DHCP fails, it will try Dynamic Link Local Addressing (DLLA). If DLLA fails, it will use the manually specified settings. When method is lan.MANUAL, only the manually specified settings will be used. Neither DHCP nor DLLA will be attempted.
Example	To display the current method: <pre>print(lan.config.method)</pre>

lan.config.speed	
Attribute	Configures LAN speed.
Usage	To read LAN speed: <pre>speed = lan.config.speed</pre> To write LAN speed: <pre>lan.config.speed = speed</pre> <p>speed: Lan speed setting in Mbps. Setting can be either 10 or 100.</p>
Remarks	This attribute selects the transmission speed used by the LAN interface when lan.config.autonegotiate (on page 13-191) is disabled. When lan.config.autonegotiate (on page 13-191) is enabled, this setting is ignored. <hr/> <p>NOTE This attribute does not indicate the actual setting currently in effect. Use the lan.status attributes to determine the current operating state of the LAN.</p> <hr/>
Also see	lan.config.autonegotiate (on page 13-191)
Example	To configure LAN speed for 100: <pre>lan.config.speed = 100</pre>

lan.config.subnetmask	
Attribute	Configures LAN subnet mask.
Usage	To read the LAN subnet mask: <pre>mask = lan.config.subnetmask</pre> To write the LAN subnet mask: <pre>lan.config.subnetmask = mask</pre> <p>mask: LAN subnet mask value string specifying the subnet mask in dotted decimal notation.</p>
Remarks	This attribute specifies the LAN subnet mask to use when the manual configuration method is used to configure the LAN. This setting is ignored when DLLA or DHCP is used. <hr/> <p>NOTE This attribute does not indicate the actual setting currently in effect. Use the lan.status attributes to determine the current operating state of the LAN.</p> <hr/>
Also see	lan.status
Example	To display the LAN subnet mask: <pre>print(lan.config.subnetmask)</pre>

lan.lxidomain	
Attribute	Sets LXI domain.
Usage	To read the LXI domain: <code>domain = lan.lxidomain</code> To write the LXI domain: <code>lan.lxidomain = domain</code> domain: The LXI domain number (0–255, default = 0).
Remarks	This attribute is used to set the LXI domain. It must be a number between 0 and 255. The default value is 0. All outgoing LXI packets will be generated with this domain number. All inbound LXI packets will be ignored unless they have this domain number.
Example	To display LXI domain: <code>print(lan.lxidomain)</code>

lan.pingenable	
Attribute	Set or read the LAN ping state.
Usage	To read the LAN ping state: <code>pingstate = lan.pingenable</code> To write the LAN ping state: <code>lan.pingenable = pingstate</code> pingstate: Enable or disable ping (0 disables, or 1 enables).
Example	To display the present LAN ping state: <code>print(lan.pingenable)</code>

lan.restoredefaults()	
Function	Reset LAN settings to their defaults.
Usage	<code>lan.restoredefaults()</code>

lan.restoredefaults()	
Remarks	<p>This function restores all LAN to their default values. The following list details what settings are restored.</p> <ul style="list-style-type: none"> • lan.autoconnect: lan.DISABLE • lan.config.autonegotiate: lan.ENABLE • lan.config.dns.address[N]: 0.0.0.0 • lan.config.dns.domain: "" • lan.config.dns.dynamic: lan.ENABLE • lan.config.dns.hostname: "" • lan.config.dns.verify: lan.ENABLE • lan.config.duplex: lan.FULL • lan.config.gateway: 0.0.0.0 • lan.config.ipaddress: 0.0.0.0 • lan.config.method: lan.AUTO • lan.config.speed: 100 • lan.config.subnetmask: 0.0.0.0 • lan.linktimeout: 20 (seconds) • lan.lxidomain: 0 • lan.nagle: lan.ENABLE • lan.timedwait: 20 (seconds) <hr/> <p>NOTE This function does not reset the web password. The localnode.password attribute controls the web password and it can be reset separately.</p>
Example	<p>To reset the LAN settings:</p> <pre>lan.restoredefaults()</pre>

lan.status.dns.address[N]	
Attribute	Reads present DNS server IP addresses.
Usage	<pre>dnsaddress = lan.status.dns.address[index]</pre> <p>index: Entry index (1, 2, or 3) dnsaddress: DNS server IP address.</p>
Remarks	This attribute is an array of DNS server addresses. Up to three addresses may be in use.

lan.status.dns.address[N]	
Details	<p>NOTE Unused or disabled entries will be returned as "0.0.0.0" when read. dnsaddress returned is a string specifying the DNS server's IP address in dotted decimal notation.</p> <p>NOTE Although only two address may be manually specified, the unit will use up to three DNS server addresses. If two are specified here, only one given by a DHCP server will be used. If no entries are specified here, up to three address given by a DHCP server will be used.</p>
Also see	lan.config.dns.domain (on page 13-192)
Example	<p>To display present DNS address 1 (if it equals 164.109.48.173):</p> <pre>print(lan.status.dns.address[1])</pre> <p>→ 164.109.48.173</p>
lan.status.dns.hostname	
Attribute	Reads present DNS fully qualified host name.
Usage	<p>To read Dynamic DNS hostname:</p> <pre>hostname = lan.status.dns.hostname</pre> <p>hostname: Fully qualified DNS hostname.</p>
Remarks	This attribute holds the fully qualified DNS host name that can be used to reach the unit. This name includes the DNS domain. If a DNS host name for the unit was not found, this attribute will hold the IP address in dotted decimal notation.
Also see	lan.config.dns.hostname (on page 13-193)
Example	<p>To display the present Dynamic DNS hostname:</p> <pre>print(lan.status.dns.name)</pre>
lan.status.duplex	
Attribute	Reads present status of LAN duplex mode.
Usage	<p>To read LAN duplex mode:</p> <pre>duplex = lan.status.duplex</pre> <p>duplex: LAN duplex setting can be one of the following values:</p> <ul style="list-style-type: none"> lan.FULL or 1: full-duplex operation. lan.HALF or 0: half-duplex operation.
Remarks	This attribute indicates which duplex mode is currently in use by the LAN interface.
Also see	lan.status
Example	<p>To display present LAN duplex mode:</p> <pre>print(lan.status.duplex)</pre> <p>→ 1.000000000e+000</p>

lan.status.gateway	
Attribute	Reads present LAN default gateway address.
Usage	gatewayaddress = lan.status.gateway gatewayaddress: LAN default gateway address.
Remarks	This attribute indicates the default gateway IP address setting currently in effect.
Also see	lan.config.gateway (on page 13-195)
Example	To display present gateway address: <pre>print(lan.status.gateway)</pre>

lan.status.ipaddress	
Attribute	Reads present LAN IP address.
Usage	To read the LAN IP address: <pre>ipaddress = lan.config.ipaddress</pre> To write the LAN IP address: <pre>lan.config.ipaddress = ipaddress</pre> ipaddress: LAN IP address specified in dotted decimal notation.
Remarks	This attribute indicates the LAN IP address currently in use.
Also see	lan.config.ipaddress (on page 13-195)
Example	To display the present LAN IP address: <pre>print(lan.status.ipaddress)</pre>

lan.status.macaddress	
Attribute	Reads LAN MAC address.
Usage	macaddress = lan.status.macaddress macaddress: The Series 3700 MAC address.
Remarks	This attribute provides the unit's LAN MAC address. The MAC address is a character string representing the unit's MAC address in hexadecimal notation. The string will have colons separating the address octets (see example).
Example	To display the Series 3700 MAC address if it is "00:60:1A:00:00:57": <pre>print(lan.status.macaddress)</pre> → 00:60:1A:00:00:57

lan.status.port.dst	
Attribute	Reads present LAN dead socket termination port number.
Usage	port = lan.status.port.dst port: Dead socket termination socket port number.

lan.status.port.dst	
Remarks	This attribute holds the TCP port number used to reset all other LAN socket connections.
Example	To display the Series 3700 DST port number: <pre>print(lan.status.port.dst)</pre> → 5.030000000e+003

lan.status.port.rawsocket	
Attribute	Reads present LAN raw socket connection port number.
Usage	<pre>port = lan.status.port.rawsocket</pre> port: Raw socket port number.
Remarks	This attribute holds the TCP port number used to connect to the instrument to control it over a raw socket communication interface.
Example	To display the Series 3700 raw socket port number: <pre>print(lan.status.port.rawsocket)</pre> → 5.025000000e+003

lan.status.port.telnet	
Attribute	Reads present LAN telnet connection port number.
Usage	<pre>port = lan.status.port.telnet</pre> port: Telnet port number.
Remarks	This attribute holds the TCP port number used to connect to the instrument to control it over a telnet interface.
Example	To display the Series 3700 TCP port number: <pre>print(lan.status.port.telnet)</pre> → 2.300000000e+001

lan.status.port.vxi11	
Attribute	Reads present LAN VXI-11 connection port number.
Usage	<pre>port = lan.status.port.vxi11</pre> port: LAN VXI-11 port number.
Remarks	This attribute holds the TCP port number used to connect to the instrument to control it over a VXI-11 protocol connection.
Example	To display the Series 3700 VXI-11 number: <pre>print(lan.status.port.vxi11)</pre> → 1.024000000e+003

lan.status.reset()	
Function	Resets the LAN interface.
Usage	<code>lan.status.reset()</code>
Remarks	This function performs a lan.restoredefaults() (on page 13-198) followed by a lan.applysettings() (on page 13-190). To restore the LAN settings without applying them, use <code>lan.restoredefaults()</code> .
Example	To reset the LAN interface: <code>lan.status.reset()</code>

lan.status.speed	
Attribute	Reads present LAN speed.
Usage	<code>speed = lan.status.speed</code> speed: LAN speed given in Mbps. It will be either 10 or 100.
Remarks	This attribute indicates the transmission speed currently in use by the LAN interface.
Example	To display the Series 3700 transmission speed presently in use: <code>print(lan.status.speed)</code> → 1.000000000e+002

lan.status.subnetmask	
Attribute	Reads present LAN subnet mask.
Usage	<code>mask = lan.status.subnetmask</code> mask: A string specifying the subnet mask in dotted decimal notation.
Remarks	This attribute indicates the LAN subnet mask currently in use.
Example	To display the Series 3700 subnet mask presently in use: <code>print(lan.status.subnetmask)</code> → 255.255.255.0

lan.trigger[N].assert()	
Function	Simulates the occurrence of the trigger and generates the corresponding event id.
Usage	<code>lan.trigger[N].assert()</code> N: The trigger packet over LAN to assert (1–8).
Remarks	This function will generate a trigger for the specified LAN packet.
Also see	lan.trigger[N].clear() (on page 13-204) lan.trigger[N].mode (on page 13-205) lan.trigger[N].overrun (on page 13-206) lan.trigger[N].stimulus (on page 13-208) lan.trigger[N].wait() (on page 13-209)

lan.trigger[N].assert()	
Example	To create a trigger with LAN packet 5: <code>lan.trigger[5].assert()</code>

lan.trigger[N].clear()	
Function	Clear the event detector for a trigger.
Usage	<code>lan.trigger[N].clear()</code> N: The trigger packet over LAN to clear (1–8).
Remarks	A trigger's event detector remembers if an event has been detected since the last <code>lan.trigger[packet].wait</code> call. This function clears a trigger's event detector and discards the previous history of the trigger packet.
Also see	lan.trigger[N].assert() (on page 13-203) lan.trigger[N].overrun (on page 13-206) lan.trigger[N].stimulus (on page 13-208) lan.trigger[N].wait() (on page 13-209)
Example	To clear the event detector with LAN packet 5: <code>lan.trigger[5].clear()</code>

lan.trigger[N].connect()	
Function	Connect the <code>lan.trigger</code> instance to the listener(s) specified by protocol and ipaddress.
Usage	<code>lan.trigger[N].connect()</code>
Example	<code>lan.trigger[1].protocol=lan.MULTICAST</code> <code>lan.trigger[1].connect()</code> <code>lan.trigger[1].assert()</code>

lan.trigger[N].connected	
Attribute	Returns true (if connected) or false (if not connected).
Usage	<code>lan.trigger[N].connected</code>
Example	<code>lan.trigger[1].protocol=lan.MULTICAST</code> <code>lan.trigger[1].connect()</code> <code>print(lan.trigger[1].connected)</code> <code>false</code>

lan.trigger[N].EVENT_ID	
Attribute	Event identifier use to route the LAN trigger to other subsystems (using stimulus properties).
Usage	<code>lan.trigger[N].EVENT_ID</code>
Example	<code>digio.trigger[14].stimulus = lan.trigger[1].EVENT_ID</code>

lan.trigger[N].ipaddress	
Attribute	Specify the address (in dotted decimal) of UDP or TCP listeners. Set to 0.0.0.0 for MULTICAST.
Usage	lan.trigger[N].ipaddress = dotted decimal (ddd.ddd.ddd.ddd)
Example	lan.trigger[3].protocol=lan.TCP lan.trigger[3].ipaddress="192.168.1.100" lan.trigger[3].connect()

lan.trigger[N].mode	
Attribute	Sets the trigger operation/detection mode.
Usage	To read the trigger operation/detection mode: mode = lan.trigger[N].mode To write the trigger operation/detection mode: mode = lan.trigger[N].mode N: The LAN event number (1-8). mode: Trigger mode (1-7).

lan.trigger[N].mode	
Remarks	<p>This attribute controls the mode in which the trigger event detector, as well as the output trigger generator, will operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers. When setting, mode can be one of the following values:</p> <ul style="list-style-type: none"> • lan.TRIG_EITHER (or 0): Detect rising or falling edge (positive or negative state) trigger packets as input. Generate a LAN trigger packet with a negative state for output. • lan.TRIG_FALLING (or 1): Detect falling edge (negative state) trigger packets as input. Generate a LAN trigger packet with a negative state for output. • lan.TRIG_RISING (or 2): Detect rising edge (positive state) trigger packets as input. Generate a LAN trigger packet with a positive state for output. • lan.TRIG_RISINGA (or 3): Same as TRIG_RISING. Detect rising edge (positive state) trigger packets as input. Generate a LAN trigger packet with a positive state for output. • lan.TRIG_RISINGM (or 4): Same as TRIG_RISING. Detect rising edge (positive state) trigger packets as input. Generate a LAN trigger packet with a positive state for output. • lan.TRIG_SYNCHRONOUS (or 5): Detect falling edge (negative state) trigger packets as input. Generate a LAN trigger packet with a positive state for output. • lan.TRIG_SYNCHRONOUSA (or 6): Detect falling edge (negative state) trigger packets as input. Generate a LAN trigger packet with a positive state for output. • lan.TRIG_SYNCHRONOUSM (or 7): Detect rising edge (positive state) trigger packets as input. Generate a LAN trigger packet with a negative state for output. <p>The default trigger mode for a trigger is TRIG_EITHER.</p> <p>TRIG_SYNCHRONOUS is provided for compatibility with the digital I/O triggering on older firmware. Use of TRIG_SYNCHRONOUSA or TRIG_SYNCHRONOUSM (instead of TRIG_SYNCHRONOUS) is preferred.</p>
Example	<p>To see the present LAN trigger mode of LAN event 1 :</p> <pre>print(lan.trigger[1].mode)</pre>

lan.trigger[N].overrun	
Attribute	Event detector overrun status.
Usage	<pre>overrun = lan.trigger[N].overrun</pre> <p>N: The trigger packet over LAN to check overrun status (1–8). overrun: The trigger overrun state for the LAN packet specified.</p>
Remarks	<p>This attribute is a read-only attribute that indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the even detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event. It also is not an indication of an output trigger overrun. Output trigger overrun indications are provided in the status model.</p>

lan.trigger[N].overrun	
Also see	lan.trigger[N].assert() (on page 13-203) lan.trigger[N].clear() (on page 13-204) lan.trigger[N].stimulus (on page 13-208) lan.trigger[N].wait() (on page 13-209)
Example	To check the overrun status of a trigger with LAN packet 5: <pre>overrun = lan.trigger[5].overrun</pre>

lan.trigger[N].protocol	
Attribute	Sets LAN protocol to use for sending trigger messages.
Usage	<p>To read present LAN protocol:</p> <pre>protocol = lan.trigger[N].protocol</pre> <p>To write present LAN protocol:</p> <pre>lan.trigger[N].protocol = protocol</pre> <p>N: The LAN event number (1-8). protocol: The protocol to use for the trigger's messages (0, 1, or 2).</p>
Remarks	<p>This attribute selects which protocol will be used for sending trigger messages. The LAN trigger will listen for trigger messages from either protocol but will use the designated protocol for sending outgoing messages. Call <code>lan.trigger[N].connect</code> after changing this setting before outgoing event messages can be sent.</p> <p>protocol must be either <code>lan.TCP</code> (or 0), <code>lan.UDP</code> (or 01), or <code>lan.MULTICAST</code> (or 2). The default value is <code>lan.TCP</code>. When the <code>lan.MULTICAST</code> protocol is selected, the <code>ipaddress</code> attribute will be ignored and event messages will be sent to the multicast address 224.0.23.159.</p>
Example	To view LAN protocol to use for sending trigger messages for LAN event 1: <pre>print(lan.trigger[1].protocol)</pre>

lan.trigger[N].pseudostate	
Attribute	Sets the simulated line state for the LAN trigger.
Usage	<p>To read the simulated line state for the LAN trigger:</p> <pre>pseudostate = lan.trigger[N].pseudostate</pre> <p>To write the simulated line state for the LAN trigger:</p> <pre>lan.trigger[N].pseudostate = pseudostate</pre> <p>N: The LAN event number (1-8). pseudostate: The simulated line state (0 or 1).</p>
Remarks	<p>This attribute tracks the simulated line state of the LAN trigger. The value can be set to initialize the pseudo state to known value. Setting this attribute will not cause the LAN trigger to generate any events or output packets. ON or OFF cannot be used when setting the pseudostate.</p>

lan.trigger[N].pseudostate	
Example	To display the present simulated line state for the LAN event 1: <pre>print(lan.trigger[1].pseudostate)</pre>
lan.trigger[N].stimulus	
Attribute	Event to cause this trigger to assert.
Usage	To read the trigger stimulus: <pre>trigstim = lan.trigger[N].stimulus</pre> packet: The trigger packet over LAN to query for stimulus setting. trigstim: The event identifier being used to trigger the event. To write the trigger stimulus: <pre>lan.trigger[N].stimulus = trigstim</pre> N: The trigger packet over LAN for which to set the trigger source (1–8). trigstim: The event identifier to set as the trigger event stimulus.
Remarks	This attribute selects which event will cause a LAN trigger packet to be sent for this trigger. The events may be one of the following: trigstim may be one of the following (existing trigger event IDs): <ul style="list-style-type: none"> • digio.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line. • display.trigger.EVENT_ID: The trigger key on the front panel is pressed. • trigger.EVENT_ID: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event. • trigger.blender[N].EVENT_ID: A combination of events has occurred. • trigger.timer[N].EVENT_ID: A delay expired. • tsplink.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line. • lan.trigger[N].EVENT_ID • scan.trigger.EVENT_SCAN_READY: Scan Ready Event. • scan.trigger.EVENT_SCAN_START: Scan Start Event • scan.trigger.EVENT_CHANNEL_READY: Channel Ready Event • scan.trigger.EVENT_MEASURE_COMP: Measure Complete Event • scan.trigger.EVENT_SEQUENCE_COMP: Sequence Complete Event • scan.trigger.EVENT_SCAN_COMP: Scan Complete Event • scan.trigger.EVENT_IDLE: Idle Event <hr/> <p>NOTE Use the ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because they may need to change when enhancements are added to the instrument.</p>

lan.trigger[N].stimulus	
Also see	lan.trigger[N].assert() (on page 13-203) lan.trigger[N].clear() (on page 13-204) lan.trigger[N].overrun (on page 13-206) lan.trigger[N].wait() (on page 13-209)
Example	To use timer 1 trigger event as the source for LAN packet 5 trigger stimulus: <pre>lan.trigger[5].stimulus = trigger.timer[1].EVENT_ID</pre>

lan.trigger[N].wait()	
Function	Wait for a trigger.
Usage	<pre>triggered = lan.trigger[N].wait(timeout)</pre> <p>N: The trigger packet over LAN to wait for (1–8).</p> <p>timeout: Maximum amount of time in seconds to wait for the trigger.</p> <p>triggered: Trigger detection indication.</p>
Remarks	<p>This function will wait for an input trigger. If one or more trigger events were detected since the last time <code>lan.trigger[N].wait</code> or <code>lan.trigger[N].clear</code> was called, this function will return immediately.</p> <p>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.</p>
Also see	lan.trigger[N].assert() (on page 13-203) lan.trigger[N].clear() (on page 13-204) lan.trigger[N].overrun (on page 13-206) lan.trigger[N].stimulus (on page 13-208)
Example	To wait for a trigger with LAN packet 5 with a timeout of 3 seconds: <pre>triggered = lan.trigger[5].wait(3)</pre>

localnode functions and attributes

Use the attributes in this group to set the power line frequency, control (on/off) prompting, and control (hide/show) error messages on the display.

localnode.define.*		.MAX_TIMERS .MAX_DIO_LINES .MAX_TSPLINK_TRIGS .MAX_BLENDERS .MAX_BLENDER_INPUTS .MAX_LAN_TRIGS
Attribute	Indicates the maximum number available for each feature.	
Usage	To read the maximum number available for a feature: maxnum = localnode.define.MAX_TIMERS maxnum = localnode.define.MAX_DIO_LINES maxnum = localnode.define.MAX_TSPLINK_TRIGS maxnum = localnode.define.MAX_BLENDERS maxnum = localnode.define.MAX_BLENDER_INPUTS maxnum = localnode.define.MAX_LAN_TRIGS	
Example	To read the maximum number of timers that are available: maxnum = localnode.define.MAX_TIMERS	
localnode.description		
Attribute	User description of the unit.	
Usage	localnode.description = description description = localnode.description description: User description of the unit.	
Remarks	This attribute holds a string with an arbitrary description of the unit. This value will appear on the welcome web page for the unit.	
localnode.execute()		
Function	Execute TSL code.	
Usage	localnode.execute(chunk) chunk: Source TSL code to execute.	

localnode.execute()	
Remarks	This function will execute the given TSL code. NOTE This command cannot actually be used on the local node. It is provided for the sole purpose of executing scripts on this node from a remote master node. The localnode prefix to the command is an artifact of command organization and how remote commands are shared between nodes.

localnode.getglobal()	
Function	Get a the value of a global variable.
Usage	value = localnode.getglobal(name) name: The global variable name. value: The value of the variable.
Remarks	This function will return the value of the given global variable. This function is provided to allow code running on a remote master node to retrieve values of variables from that node. This function should not be used to retrieve the value of a global variable on the local node when using the local node as the master. Accessing the variable directly is far more efficient. NOTE This command is provided for the sole purpose of accessing variables on this node from a remote master node. The localnode prefix to the command is an artifact of command organization and how remote commands are shared between nodes.

localnode.linefreq	
Attribute	Use to read power line frequency detected at power up.
Usage	To read line frequency: frequency = localnode.linefreq
Remarks	<ul style="list-style-type: none"> 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.
Example	To get the line frequency: print(localnode.linefreq)

localnode.model	
Attribute	Use to query model number.
Usage	value = localnode.model value: Represents model number.

localnode.model	
Example	To print model number: <pre>print(localnode.model)</pre> Output: 3706

localnode.password	
Attribute	The remote access password.
Usage	<code>localnode.password = "password"</code>
Remarks	This attribute holds the remote access password. When password usage is enabled, this password must be supplied to change the configuration or control a unit from a web page or a remote command interface. This attribute is write-only and cannot be read. NOTE The unit will continue to use the old password for all interactions until the command to change it executes. When changing the password, give the unit time to execute the command before attempting to use the new password.

localnode.passwordmode	
Attribute	The remote access password enable mode.
Usage	<code>localnode.passwordmode = mode</code> <code>mode = localnode.passwordmode</code> mode: The password enable mode.
Remarks	This attribute controls if and where remote access passwords are required. Set this attribute to one of the values in the table below to enable password checking. <code>localnode.PASSWORD_NONE</code> : Disable passwords everywhere. <code>localnode.PASSWORD_WEB</code> : Use passwords on the web interface only. <code>localnode.PASSWORD_LAN</code> : Use passwords on the web interface and all Ethernet interfaces. <code>localnode.PASSWORD_ALL</code> : Use passwords on the web interface and all remote command interfaces.

localnode.prompts	
Attribute	Prompting mode.

localnode.prompts	
Usage	<p>To read prompting state:</p> <pre>prompting = localnode.prompts</pre> <p>To write prompting state:</p> <pre>localnode.prompts = prompting</pre> <p>prompting: Set to 0 to disable or 1 to enable.</p>
Remarks	<ul style="list-style-type: none"> • 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 3700 generates prompts in response to command messages. • When the prompting mode is enabled, the Series 3700 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. When downloading scripts, 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 requires prompts. It sets the prompting mode behind the scenes. If you disable prompting, use of the Test Script Builder will hang because it will be waiting for the prompt that lets it know that the command is done executing. DO NOT disable prompting with the use of the Test Script Builder. • When used in an expanded system (TSP-Link™), localnode.prompt is sent to the Remote Master node only. Use <code>node [N] .prompt</code> (where N is the node number) to send the command to any node in the system.
Also see	localnode.showerrors (on page 13-217)
Example	<p>Enables prompting:</p> <pre>localnode.prompts = 1</pre>
localnode.reset()	
Function	Resets the system.
Usage	<code>localnode.reset()</code>

localnode.reset()	
Remarks	<p>A system reset includes a <code>channel.reset('allslots')</code>, <code>dmm.reset('all')</code>, and a <code>scan.reset()</code>. In addition:</p> <ul style="list-style-type: none"> • Other system settings are restored back to factory default settings. • Existing channel patterns and DMM configurations are deleted. • All channels and backplane relays open. • The dmm function is "dcvolts". • User-created reading buffers are deleted.
Also see	<p>channel.reset() (on page 13-68)</p> <p>dmm.reset() (on page 13-161)</p> <p>scan.reset() (on page 13-242)</p>

localnode.revision	
Attribute	Use to query the firmware revision level.
Usage	<p><code>value = localnode.revision</code></p> <p>value: Firmware revision level.</p>
Example	<p>To output the present revision level:</p> <pre>print(localnode.revision)</pre> <p>Output:</p> <p>01.00a</p>

localnode.serialno	
Attribute	Use to query serial number:
Usage	<p>To read serial number:</p> <p><code>value = localnode.serialno</code></p> <p>value: Series 3700's serial number</p>
Example	<p>To output the unit's serial number:</p> <pre>print(localnode.serialno)</pre> <p>Output:</p> <p>01116353</p>

localnode.setglobal()	
Function	Set a the value of a global variable.
Usage	<p><code>localnode.setglobal(name, value)</code></p> <p>name: The global variable name to create.</p> <p>value: The value to assign to the variable.</p>

localnode.setglobal()	
Remarks	<p>This function will assign the given value to a global variable. This function is provided to assign values to variables from a remote master node. This function should not be used to assign values to global variables on the local node when using the local node as the master, assigning the value directly is far more efficient.</p> <hr/> <p>NOTE This command is provided for the sole purpose of accessing variables on this NOTE from a remote master node. The <code>localnode</code> prefix to the command is an artifact of command organization and how remote commands are shared between nodes.</p>

localnode.settime()	
Function	Set the current time of the system.
Usage	<pre>settime(hour, minute, second)</pre> <p>or</p> <pre>localnode.settime(os.time(year = <year>, month = <month>, day = <day>, hour = <hour>, min = <min>, sec = <sec>))</pre> <p><year>: A full year that is 2006 or later <month>: The desired month from 01 to 12 <day>: The desired day from 01 to 31 <hour>: The desired hour from 00 to 23 <minute>: The desired minute from 00 to 59 <second>: The desired second from 00 to 59</p>
Remarks	<p>This function sets the date and time of the system based on the <code>os.time</code> response passed in as its parameter. Use year, month, day, hour, min, and sec to set the time as desired. The first 3 parameters to <code>os.time</code> are mandatory while the rest are optional. If the later 3 are not used, they default to noon for that day. The setting of the time and date does not take into account the time zone. Please update the time for your time zone.</p>
Example	<p>To set the date and time to Oct 3, 2006 at 2:25 pm:</p> <pre>settime(os.time(year = 2006, month = 10, day = 3, hour =14, min = 25, sec = 0))</pre>

localnode.setup.poweron	
Attribute	The saved setup to recall when the unit is turned on.
Usage	<p>To read power on state:</p> <pre>n = localnode.setup.poweron</pre> <p>n: Returned power on state.</p> <p>To write power on state:</p> <pre>localnode.setup.poweron = n</pre> <p>n: Setup number to recall on power up (0 or 1).</p>

localnode.setup.poweron	
Remarks	Setting this attribute to 0 causes the unit to power up to the factory default (reset) setup. A setting of 1 causes the unit to power up using a user setup that was previously saved internally.
Example	To set unit to power on with factory default settings: <pre>localnode.setup.poweron = 0</pre> To query power on state: <pre>print(localnode.setup.poweron)</pre> Output (factory default): 0

localnode.setup.recall()	
Function	Recalls settings from a saved setup.
Usage	<code>localnode.setup.recall(location)</code> location: Setup number to recall (0, 1, or "/usb1/<filename>"). 0: Reset setup. 1: Internal setup. <filename>: Use the name of the desired file contained on a USB flash drive.
Remarks	If a number is sent as the parameter: The number is interpreted as a setup number and the setup is recalled from internal memory. Setting this attribute to 0 recalls the factory default (reset) setup. Setting this attribute to 1 recalls the user saved setup from internal memory. If a string is sent as the parameter: The string is interpreted as a path and filename and the setup is recalled from the corresponding file on the USB flash drive. The path may be absolute or relative to the current working directory. The filename must include the file extension.
Also see	localnode.setup.save() (on page 13-216)
Example	To recall factory default settings: <pre>localnode.setup.recall(0)</pre> To recall the user saved setup from internal memory: <pre>localnode.setup.recall(1)</pre> To recall a user saved setup stored in a file named KEITHLEY_3730 on a USB flash drive: <pre>localnode.setup.recall("/usb1/KEITHLEY_3730.set")</pre>

localnode.setup.save()	
Function	Saves the present setup as the user-setup.

localnode.setup.save()	
Usage	<p>To save to the internal memory location, send no parameters with function:</p> <pre>localnode.setup.save()</pre> <p>To save to the USB flash drive:</p> <pre>localnode.setup.save(location)</pre> <p>location: Setup location to save. Use the format "/usb1/<filename>" where <filename> is the name of the desired file contained on a USB flash drive. The location must include the /usb1/. If it includes a file extension, it must be .set. If no extension is provided, .set will be appended automatically to filename. The .set file extension is used on front panel to identify setup files to load from thumb drive.</p>
Remarks	This function overwrites any previous values with the present setup.
Also see	localnode.setup.recall() (on page 13-216)
Example	<p>To save the present setup as the internal user setup:</p> <pre>localnode.setup.save()</pre> <p>To save a user saved setup to a file named KEITHLEY_3730 on a USB flash drive:</p> <pre>localnode.setup.save("/usb1/KEITHLEY_3730")</pre>
localnode.showerrors	
Attribute	Automatic display of errors.
Usage	<p>To read the show errors state:</p> <pre>errormode = localnode.showerrors</pre> <p>To write the show errors state:</p> <pre>localnode.showerrors = errormode</pre> <p>errormode: Set to 0 or 1.</p>
Remarks	<ul style="list-style-type: none"> • If this attribute is set to 1, for any errors that are generated, the unit will automatically display the errors stored in the error queue, and then clear the queue. Errors will be processed at the end of executing a command message (just prior to issuing a prompt if prompts are enabled). • 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.
Details	See errorqueue functions and attributes (on page 13-176).
Also see	localnode.prompts (on page 13-212)
Example	<p>Displays errors:</p> <pre>localnode.showerrors = 1</pre>

makegetter functions

Use the functions in this group to set and retrieve a value for an attribute.

makegetter()	
Function	Creates a function to get the value of an attribute.
Usage	<pre>getter = makegetter(table, attributename)</pre> <p>table: Read-only table where the attribute is located. attributename: The string name of the attribute. getter: Function that returns the value of the given attribute.</p>
Remarks	<ul style="list-style-type: none"> 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.
Also see	makesetter() (on page 13-218)
Example	<p>To create a getter function called <code>getrange</code>:</p> <pre>getrange = makegetter(dmm, "range") ... r = getrange()</pre> <hr/> <p>NOTE When <code>getrange</code> is called, it returns the value of <code>dmm.range</code>.</p>

makesetter()	
Function	Creates a function to set the value of an attribute.
Usage	<pre>setter = makesetter(table, attributename)</pre> <p>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.</p>
Remarks	<ul style="list-style-type: none"> 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.
Also see	makegetter() (on page 13-218)
Example	<p>Use <code>setrange</code> to set the value of <code>dmm.range</code> for the currently selected function:</p> <pre>setrange = makesetter(dmm, "range") setrange(5)</pre>

memory functions

memory.available()	
Function	Indicates the memory available in the system.
Usage	<pre>mem_avail = memory_available()</pre> <p>mem_avail: Comma-delimited string with percentages for available memory.</p> <p>The string format of mem_avail is "sys_mem, script_mem, pat_mem, config_mem", where:</p> <p>sys_mem: Percentage of overall memory in the system</p> <p>script_mem: Percentage of memory available in the system to store user scripts</p> <p>pat_mem: Percentage of memory available in the system to store channel patterns</p> <p>config_mem: Percentage of memory available in the system to store user DMM configurations</p>
Remarks	Use this function to view the available memory in the system overall as well as the memory available for storing user scripts, for storing channel patterns, and for storing user DMM configurations. The response to this function is a single string that provides the overall system memory available as well as the script memory available, channel pattern memory available, and DMM configuration memory available as comma-delimited percentages.
Also see	<code>memory.used()</code> (on page 13-219)
Example	<p>To read the memory available in the system:</p> <pre>MemAvail = memory.available()</pre> <p>To print out the memory in the system:</p> <pre>print(MemAvail)</pre> <p>or</p> <pre>print(memory.available())</pre> <p>Output: 51.56, 92.84, 100.00, 100.00 00</p> <p>After recalling a setup that was saved internally:</p> <pre>setup.recall(1)</pre> <pre>print(memory.available()) -> 11.13, 92.84, 0.16, 97.03</pre>
memory.used()	
Function	Indicates the memory available in the system.

memory.used()	
Usage	<pre>mem_avail = memory_available()</pre> <p>mem_avail: Comma-delimited string with percentages for available memory.</p> <p>The string format of mem_avail is "sys_mem, script_mem, pat_mem, config_mem", where:</p> <p>sys_mem: Percentage of overall memory in the system</p> <p>script_mem: Percentage of memory available in the system to store user scripts</p> <p>pat_mem: Percentage of memory available in the system to store channel patterns</p> <p>config_mem: Percentage of memory available in the system to store user DMM configurations</p>
Remarks	Use this function to view the used memory in the system overall as well as the memory used for storing user scripts, for storing channel patterns, and for storing user DMM configurations. The response to this function is a single string that provides the overall system memory used as well as the script memory available, channel pattern memory available, and DMM configuration memory used as comma-delimited percentages.
Also see	memory.available() (on page 13-219)
Example	<p>To read the memory used in the system:</p> <pre>MemUsed = memory.used()</pre> <p>To print out the memory used in the system:</p> <pre>print(MemUsed)</pre> <p>or</p> <pre>print(memory.used())</pre> <p>Output: 48.44, 7.16,0.00, 0.00</p> <p>After recalling a setup that was saved internally:</p> <pre>setup.recall(1)</pre> <pre>print(memory.available()) -> 88.87. 7.16, 99.84, 2.97</pre>

opc functions

Use this function to set 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	<code>opc()</code>

opc()	
Remarks	<ul style="list-style-type: none"> • 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.
Also see	waitcomplete (on page 3-16)

print functions

Use these functions to print.

printbuffer()	
Function	Prints data from reading buffer.
Usage	<p>There are multiple ways to use this function, depending on how many sub-tables are used:</p> <pre>printbuffer(start_index, end_index, st_1) printbuffer(start_index, end_index, st_1, st_2) printbuffer(start_index, end_index, st_1, st_2, ..., st_n)</pre> <p>start_index: Starting index of values to print. end_index: Ending index of values to print. st_1, st_2, ... st_n: Sub-tables from which to print values.</p>

printbuffer()	
Remarks	<ul style="list-style-type: none"> • Correct usage when there are no outstanding overlapped commands to acquire data: • $1 \leq \text{start_index} \leq \text{end_index} \leq n$ Where n refers to the index of the last entry in the tables to be printed. • If $\text{end_index} < \text{start_index}$ or $n < \text{start_index}$, no data will be printed. If $\text{start_index} < 1$, 1 will be used as the first index. If $n < \text{end_index}$, 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. • This function prints values from reading buffers. If a specific sub-table is not specified (for example, "rb1" rather than "rb1.statuses"), the default "readings" sub-table will be used. • At least one sub-table must be specified. There is an upper limit that is dictated by the output format and the maximum output message length. Values will be interleaved in one message. Care must be taken not to exceed the maximum output message length. • All the data will be put in one response message. The response message will be formatted as dictated by <code>format.data</code> and other associated attributes.
Also see	format.data (on page 13-184)
Example	<p>This example prints the readings (<code>buf</code>), the units (<code>buf.units</code>), and relative time stamps (<code>buf.relativetimestamps</code>) for the 1st and 2nd readings in the buffer named <code>buf</code>:</p> <pre>printbuffer(1,2,buf, buf.units, buf.relativetimestamps) 3.535493836e-002, Volts DC, 0.000000000e+000 -4.749810696e-002, Volts DC, 5.730966000e-002</pre>
printnumber()	
Function	Prints numbers using the format selected for printing reading buffers.
Usage	<p>There are multiple ways to use this function, depending on how many numbers are to be printed:</p> <pre>printnumber(v1) printnumber(v1 ,v2) printnumber(v1 ,v2, ..., vn)</pre> <p>v1, v2, ..., vn: Numbers to print.</p>

printnumber()	
Remarks	<ul style="list-style-type: none"> This function will print the given numbers using the data format specified by <code>format.data</code> 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() (on page 13-221) format.data (on page 13-184)
Example	Prints three measurements that were previously performed: <pre>format.data = format.ASCII printnumber(i, v, t)</pre> Example of returned data (i, v, t): 1.02345E-04, 8.76542E-02, 5.29372E-01

ptp functions and attributes

Use these functions to configure the IEEE-1588 Precision Time Protocol (PTP). IEEE-1588 allows multiple devices to synchronize time to a less than 10mS accuracy. Further information on the protocol, operation, and terminology is available from the IEEE organization documentation or other third-party sources.

ptp.burst.enable()	
Function	Enables a special mode for bursts of 1588 packets used to speed up synchronization.
Usage	<pre>ptp.burst.enable [0 1 ptp.ON ptp.OFF]</pre> <p>ptp.ON (1) ptp.OFF (0)</p>
Example	<pre>ptp.burst.enable=1 print(ptp.burst.enable)</pre>

ptp.ds.current()	
Function	Read-only string that specifies the current delay and offset timing for the instrument. If this instrument is the master, the values will be zero.
Usage	<pre>ptp.ds.current()</pre>

ptp.ds.current()	
Details	<p>The fields are:</p> <ul style="list-style-type: none"> • Steps removed – the number of steps to reach a master. 1 without any boundary clocks, >1 if one or more boundary clocks are involved • Offset from Master: the current time offset, in seconds, from the master • One Way Delay, the current one way network delay (master to slave or slave to master) • Master to Slave Delay, the current master to slave network delay, in seconds • Slave to Master Delay, the current slave to master network delay in seconds
Example	<pre>print(ptp.ds.current()) Steps removed: 1 Offset from Master: 0.000000012 One Way Delay: 0.000015794 Master to Slave Delay: 0.000015806 Slave to Master Delay: 0.000015709</pre>

ptp.ds.default()	
Function	Read-only string that specifies various characteristics of the current clock.
Usage	<code>ptp.ds.default()</code>
Example	<pre>print(ptp.ds.default()) Clock Communication: 1 Clock Port Field: 0 Clock Stratum: 255 Clock Variance: 0 Clock Followup Capable: 1 Preferred: 0 Initializable: 1 External Timing: 0 Is Boundary Clock: 0 Sync Interval: 1 Number of Ports: 1 Num of Foreign Records: 10 Clock Identifier: DFLT Clock UUID: 00 60 0c 01 83 ee Subdomain Name: _DFLT</pre>

ptp.ds.foreignmaster()	
Function	Read-only string describing information about any "foreign" clocks the instrument knows about.
Usage	<code>ptp.ds.foreignmaster()</code>
Example	<code>print(ptp.ds.foreignmaster())</code> Empty.

ptp.ds.globaltime()	
Function	Read-only string describing global properties such as UTC offset.
Usage	<code>ptp.ds.globaltime()</code>
Example	<code>print(ptp.ds.globaltime())</code> Current UTC Offset: 32 Leap 59: 0 Leap 61: 0 Epoch number: 0

ptp.ds.parent()	
Function	Read-only string describing characteristics of the current parent clock.
Usage	<code>ptp.ds.parent()</code>

ptp.ds.parent()	
Example	<pre>print(ptp.ds.parent()) Parent Communication: 1 Parent Port Id: 1 Parent Last Sync Seq: 16140 Parent Followup Capable: 1 Parent External Timing: 0 Parent Variance: -12000 Parent Stats: 0 Observed Variance: 0 Observed Drift: 0 UTC Reasonable: 0 GM Communication: 1 GM Port Id: 1 GM Stratum: 4 GM Variance: -12000 GM Preferred: 0 GM is Boundary Clock: 0 GM Sequence: 16140 Parent UUID: 00 30 d3 09 f8 b2 GM UUID: 00 30 d3 09 f8 b2 GM Identifier: DFLT</pre>

ptp.ds.portconfig()	
Function	Read-only string with information about the clock configuration, including sequence numbers, port addresses, subdomain address, and UUID.
Usage	<code>ptp.ds.portconfig()</code>

ptp.ds.portconfig()	
Example	<pre>print(ptp.ds.portconfig()) Last Sync Event Seq: 22 Last General Event Seq: 0 Port Communication: 1 Event Port Address: 319 General Port Address: 320 Port Id Field: 1 Burst Enabled: 0 Subdomain Address: 224 0 1 129 Port UUID: 00 60 0c 01 83 ee Random Number r: 29 Random Number q: 12 Sync Counter: 20 Number of Bursts: 6 Burst Counter: 0</pre>

ptp.enable()	
Function	Enable/disable 1588 on the Series 3700.
Usage	<pre>ptp.enable [0 1 ptp.ON ptp.OFF] ptp.ON (1) ptp.OFF (0)</pre>
Example	<pre>ptp.enable=1 print(ptp.enable)</pre>

ptp.portstate	
Attribute	Read-only value that indicates the state of the 1588 engine: initializing, faulting, listening, master, slave, uncalibrated, passive, or disabled (see ptp constants).
Usage	<code>ptp.portstate</code>

ptp.portstate	
Remarks	<p>ptp.INITIALIZING (0)</p> <p>ptp.FAULTY (1)</p> <p>ptp.DISABLE (2)</p> <p>ptp.LISTENING (3)</p> <p>ptp.PRE_MASTER (4)</p> <p>ptp.MASTER (5)</p> <p>ptp.PASSIVE (6)</p> <p>ptp.UNCALIBRATED (7)</p> <p>ptp.SLAVE (8)</p> <p>ptp.UNKNOWN (9)</p>
Example	<code>print (ptp.portstate)</code>

ptp.preferredmaster.enable()	
Function	<p>Indicates to 1588 if this unit wants to be a master (if all other qualifiers are equal).</p> <p>NOTE: This value is not persisted through a power cycle.</p>
Usage	<p><code>ptp.preferredmaster.enable [0 1 ptp.ON ptp.OFF]</code></p> <p>ptp.ON (1)</p> <p>ptp.OFF (0)</p>
Example	<p><code>ptp.preferredmaster.enable=1</code></p> <p><code>print (ptp.preferredmaster.enable)</code></p>

ptp.subdomain	
Attribute	Only instruments in the same subdomain will interact with each other (from a 1588 point of view).
Usage	<p><code>ptp.subdomain = [any 5 character identifier]</code></p> <p>subdomain: a qualifier used to group instruments together. The default setting is <code>_DFLT</code>.</p>
Remarks	ptp.subdomain does not take effect until 1588 is restarted (generally using a power cycle).
Example	<p><code>ptp.subdomain='_DFLT'</code></p> <p><code>print (ptp.subdomain)</code></p>

ptp.synchronized	
Attribute	Read-only value that indicates if the 1588 engine is in MASTER or SLAVE state. It does NOT indicate the PLL is settled.
Usage	<code>ptp.synchronized</code>
Example	<code>print (ptp.synchronized)</code>

ptp.syncinterval	
Attribute	Defines the interval between synchronization messages from the master. This must be set to the same value for all instruments participating in 1588 (for a given subdomain), both master and slave.
Usage	<code>ptp.syncinterval = [0,1,3,4,6]</code> 0: 1 sec 1: 2 sec 3: 8 sec 4: 16 sec 6: 64 sec
Remarks	<code>ptp.syncinterval</code> does not take effect until 1588 is restarted (generally using a power cycle).
Example	<code>ptp.syncinterval=1</code> <code>print (ptp.syncinterval)</code>

ptp.time	
Attribute	Read-only PTP time that returns seconds and fractionalseconds.
Usage	<code>ptp.time</code>
Example	<code>sec, fraction=ptp.time ()</code> <code>print (sec+fraction)</code>

ptp.utcoffset	
Attribute	Current offset, in seconds, between UTC and PTP. If the instrument is a slave, this value comes from the master (so setting the value will be overwritten on the next synchronization). The Series 3700 does not keep track of this value through a power cycle (that is, it defaults to 0 if the 3700 is the master).
Usage	<code>ptp.utcoffset</code>
Remarks	The Series 3700 is not time-zone aware, so UTC time will be presented as the local time.
Details	UTC Time = PTP Time - UTC Offset
Example	<code>ptp.utcoffset=33</code> <code>print (ptp.utcoffset)</code>

reset functions

Use this function to return all logical instruments to the default settings.

reset()	
Function	Resets the logical instruments to the default settings.
Usage	<code>reset()</code>
Remarks	<p>This function resets all logical instruments in the system. It is equivalent to iterating over all the logical instruments in the system and calling the reset method of each.</p> <p>This function is equivalent to system wide reset and includes:</p> <ul style="list-style-type: none"> • <code>channel.reset</code> • <code>dmm.reset('all')</code> • <code>scan.reset.</code> <p>It also deletes existing channel patterns and DMM configurations along with channel labels and user created reading buffers.</p>

scan functions and attributes

Use the functions in this group to specify and configure channels and/or channel patterns to scan, as well as associated buffers, triggers, or other scanning aspects.

scan.abort()	
Function	Aborts a running scan
Usage	<code>scan.abort()</code>
Remarks	This function will abort a running scan. If no scan is running, it will be ignored.

scan.add()	
Function	Add scan step to the scan list.
Usage	<code>scan.add(<ch_list>, [<dmm_config>])</code> <code>scan.add(<ch_list>, [<width>])</code> <p>ch_list: String specifying channels to add, using normal channel list syntax.</p> <p>dmm_config: Optional string listing the DMM configuration to use with items in <code>ch_list</code>.</p> <p>width: Optional value that specifies the width of the channel read.</p>

scan.add()	
Remarks	<p>Use this function to specify additional channels and channel patterns to scan. These items are appended to the end of the existing scan list that was specified by the <code>scan.create()</code> (on page 13-234) command. The items in <code>ch_list</code> are appended and scanned in the order specified in the parameter list. Specifying a channel list results in multiple steps being added to the scan.</p> <p>If the optional <code>dmm_config</code> parameter is not specified, the configuration associated with that channel or channel pattern is used (see <code>dmm.setconfig()</code> (on page 13-168) and <code>dmm.getconfig()</code> (on page 13-139)). If a <code>dmm_config</code> is specified, then that configuration is used for each channel or channel pattern specified in a temporary override mode. It does not modify the assigned configuration of a channel or channel list.</p> <p>The scan list of channels or channel patterns are not updated if any error occurs during processing of the command. However, steps already added to the scan list with prior commands remain unchanged with an error detection.</p> <p>For digital I/O or totalizer channels, the created scan step instructs the scan to read the given channel and deposit the read value into the specified reading buffer. If no reading buffer is specified, the channel is read, but the value is lost.</p> <p>The <code>width</code> parameter is only valid for channels of type digital I/O. Only a width of 1, 2, 3, or 4 is supported. If specified, the scan reads up to four consecutive channels simultaneously during the scan and deposit the resulting value into the specified reading buffer.</p> <p>DAC channels are not supported.</p> <hr/> <p>NOTE Because reading buffer time stamps are generated differently for different channel types, there can be some variance when comparing measurement time stamps to channel read time stamps.</p>
Also see	scan.create() (on page 13-234)
Example	<p>To clear the old scan list and to create a new scan list with Channels 1 to 10 on Slot 3 with <code>myDCV</code>, a user DC volts configuration, on all 10 channels and then use <code>my2wire</code>, a user 2-wire ohms configuration, on all 10 channels:</p> <pre>scan.create('3001:3010', 'myDCV') scan.add('3001:3010', 'my2wire')</pre> <p>To clear the old scan list and to create a new scan list with Channels 1 to 10 on Slot 3 with <code>myDCV</code>, a user DC volts configuration, on all 10 channels and then <code>my2wire</code>, a user 2-wire ohms configuration, on each step:</p> <pre>scan.create('') for chan = 3001, 3010 do scan.create('' .. chan, 'myDCV') scan.add('' .. chan, 'my2wire') end</pre> <hr/> <p>NOTE With respect to the <code>scan.add</code> function in the above example, the first parameter (<code>'' .. chan</code>), converts the <code>chan</code> number to a string.</p>

scan.addwrite()	
Function	Writes a specified value to a channel at the added step in the scan.
Usage	<pre>scan.addwrite(<ch_list>, <write value>, [<width>])</pre> <p>ch_list: String specifying channels to add, using normal channel list syntax.</p> <p>write_value: The value to write to the channel for this scan step.</p> <p>width: Optional value that specifies the width of the channel write.</p>
Remarks	<p><code>scan.addwrite()</code> is similar to issuing <code>channel.write()</code> at the scan step.</p> <p>Specifying a channel list results in multiple steps being added to the scan with the same <code>write</code> value.</p> <p>For digital I/O channels, only a width of 1, 2, 3, or 4 is supported. Any information (bits) greater than the specified width are ignored. Values written to inputs are ignored. If no specified channel is set for output, then an error is generated. If a width crosses channels, then only the channels set to output are affected.</p> <p>For totalizer, backplane, and switch channels, there is no valid behavior. Calling on a specific channel generates an error.</p> <p>For DAC channels, if the channel mode is changed after the scan is created, the scan is rebuilt. If the write value is no longer compatible with the new mode, an error is generated and the scan becomes invalid.</p>

scan.background()	
Function	Starts a scan and runs the scan in the background.
Usage	<pre>state, scancount, stepcount, reading = scan.background(rb_buffer)</pre> <p>rb_buffer: Optional. Reading buffer to use during scanning to store the readings. If not specified, no readings are stored during the scan.</p> <p>state: The result of scanning:</p> <ul style="list-style-type: none"> • scan.EMPTY or 0 • scan.BUILDING or 1 • scan.RUNNING or 2 • scan.ABORTED or 3 • scan.FAILED or 4 • scan.FAILED_INIT or 5, • scan.SUCCESS or 6 <p>scancount: is current scan count completed</p> <p>stepcount: is current step count completed</p> <p>reading: is the last reading of scan completed if measurements are taken during the scan.</p>

scan.background()	
Remarks	<p>This command may specify the reading buffer to use during scanning and the scan is executed in the background. The reading buffer, if specified, will store the readings and accompanying attributes desired for the scan. This command starts the scan. Prior to using this command use <code>scan.create</code> and <code>scan.add</code> to setup scan elements.</p> <p>Because scan is running in the background, use scan.state (on page 13-243) function to see current status of scanning.</p> <p>An error will be generated if the reading buffer does not exist or the parameter is not a reading buffer.</p>
Also see	<p>scan.add() (on page 13-230)</p> <p>scan.create() (on page 13-234)</p> <p>scan.execute() (on page 13-236)</p> <p>scan.list() (on page 13-237)</p> <p>scan.state() (on page 13-243)</p>
Example	<p>To use reading buffer rbuffer1 and run scan in background:</p> <pre>scan.background(rbuffer1)</pre>

scan.bypass	
Attribute	Indicates whether the first channel of the scan should wait for the channel stimulus event to be satisfied before closing
Usage	<p>To read the bypass state:</p> <pre>bypass = scan.bypass</pre> <p>To write the state of the bypass:</p> <pre>scan.bypass = bypass</pre> <p>bypass: The state of the bypass. Set bypass to one of the following values:</p> <ul style="list-style-type: none"> • scan.OFF or 0: Bypass disabled • scan.ON or 1: Bypass enabled (default)
Remarks	<p>When scan.bypass is ON (default), once the scan.trigger.arm.stimulus (on page 13-244) is satisfied, the first channel of the scan closes regardless of the scan.trigger.channel.stimulus (on page 13-246) settings. For channels other than the first, the channel stimulus must be satisfied before the channel action takes place.</p> <p>With bypass OFF, every channel (including the first) needs to have the scan.trigger.channel.stimulus (on page 13-246) settings satisfied before the channel action occurs for that step.</p>
Also see	<p>scan.trigger.arm.stimulus (on page 13-244)</p> <p>scan.trigger.channel.stimulus (on page 13-246)</p>
Example	<p>To display the present bypass state:</p> <pre>print(scan.bypass)</pre> <p>To disable the bypass option for scanning:</p> <pre>scan.bypass = scan.OFF</pre>
scan.create()	
Function	Creates a list of channels and/or channel patterns to scan.
Usage	<pre>scan.create(<ch_list>, dmm_config)</pre> <p>ch_list: A string listing the channels and/or channel patterns to replace existing scan list.</p> <p>dmm_config: Optional string listing DMM configuration to use with items in ch_list.</p>

scan.create()	
Remarks	<p>Use this function to replace an existing list of channels and/or channel patterns to scan. The existing scan list is lost after this command. These items purge the old list and start a new scan list. The items in <code>ch_list</code> will be scanned in the order specified in the parameter list.</p> <p>If the optional <code>dmm_config</code> parameter is not specified, the configuration (dmm.setconfig() (on page 13-168), dmm.getconfig() (on page 13-139)) associated with that channel or channel pattern will be used. However, if a <code>dmm_config</code> is specified, that configuration is used for each channel or channel pattern specified in a temporary override mode. It does not modify the assigned configuration of a channel or channel list.</p> <p>An error will occur if:</p> <ul style="list-style-type: none"> • A specified channel does not exist. • A specified channel pattern does not exist (slot empty or not on card). • A syntax error exists in parameter string. • An empty parameter string is specified for <code>dmm_config</code> parameter. • A specified DMM configuration does not exist. • An analog backplane relay is specified. • A forbidden channel is specified. <p>Even with an error, the scan list of channels or channel patterns will be cleared. Therefore, if the scan list is queried after sending a <code>scan.create</code> command, it will be empty if an error occurred or the specified new list if no error detected.</p> <p>Factory default is an empty scan list of channels and DMM configurations. The function scan.reset() (on page 13-242) clears the list.</p> <p>To clear the list only, send an empty string for the <code>ch_list</code> parameter.</p>
Also see	scan.add() (on page 13-230)

scan.create()	
Example	<p>To clear the old scan list without resetting the entire scan configuration aspects:</p> <pre>scan.create("")</pre> <p>To clear the old scan list and to create a new scan list with Channels 1 to 10 on Slot 3 with myDCV, a user DC volts configuration, on all 10 channels and then my2wire, a user 2-wire ohms configuration, on all 10 channels:</p> <pre>scan.create('3001:3010', 'myDCV') scan.add('3001:3010', 'my2wire')</pre> <p>To clear the old scan list and to create a new scan list with Channels 1 to 10 on Slot 3 with myDCV, a user DC volts configuration, on all 10 channels and then my2wire, a user 2-wire ohms configuration, on each step:</p> <pre>scan.create('') for chan = 3001, 3010 do scan.add('' .. chan, 'myDCV') scan.add('' .. chan, 'my2wire') end</pre> <hr/> <p>NOTE With respect to the <code>scan.add</code> function in the above example, the first parameter (<code>'' .. chan</code>), converts the <code>chan</code> number to a string.</p>

scan.execute()	
Function	Starts a scan and runs it in immediate mode.
Usage	<pre>state, scancount, stepcount, reading = scan.execute(rb_buffer)</pre> <p>rb_buffer: Optional reading buffer to use during scanning to store the readings. If not specified, no readings are stored during the scan.</p> <p>state: The result of scanning:</p> <ul style="list-style-type: none"> • scan.EMPTY or 0 • scan.BUILDING or 1 • scan.RUNNING or 2 • scan.ABORTED or 3 • scan.FAILED or 4 • scan.FAILED_INIT or 5, • scan.SUCCESS or 6 <p>scancount: is current scan count completed</p> <p>stepcount: is current step count completed</p> <p>reading: is the last reading of scan completed, if measurements were taken during the scan</p>

scan.execute()	
Remarks	<p>This command may specify the reading buffer to use during scanning and runs the scan in immediate mode. The reading buffer, if specified, will store the readings and accompanying attributes desired for the scan. This command starts the scan. Prior to using this command use scan.create() (on page 13-234) and scan.add() (on page 13-230) to setup scan elements.</p> <p>The command will not exit execution until scanning completes or is aborted by the user.</p> <p>An error will be generated if the reading buffer does not exist or the parameter is not a reading buffer.</p>
Also see	<p>scan.add() (on page 13-230)</p> <p>scan.background() (on page 13-232)</p> <p>scan.create() (on page 13-234)</p> <p>scan.list() (on page 13-237)</p> <p>scan.state() (on page 13-243)</p>
Example	<p>To use reading buffer rbuffer1 and run scan in immediate mode:</p> <pre>scan.execute(rbuffer1)</pre>

scan.list()	
Function	Use to query the existing scan list.
Usage	<pre>MyScanList = scan.list()</pre> <p>MyScanList: a string listing the existing scan step information.</p>

scan.list()	
Remarks	<p>This command will list out the existing scan list.</p> <p>If the scan list is empty, then the string "Empty Scan" is returned. Otherwise, the string will list each step in the scan along with its information for step, open, close, measure configuration, and count. For example, an existing scan list may appear as follows:</p> <pre>Init) OPEN... 1) STEP: 2007 CLOSE: 2007 MEASURE: nofunction COUNT: 1 2) STEP: 2008 OPEN: 2007 CLOSE: 2008 MEASURE: nofunction COUNT: 1 3) STEP: 2020 OPEN: 2008 CLOSE: 2020 2911 MEASURE: dcvolts COUNT: 1 4) STEP: 2021 OPEN: 2020 2911 CLOSE: 2021 2921 MEASURE: dcvolts COUNT: 1 5) STEP: 2016 OPEN: 2021 2921 CLOSE: 2016 2911 MEASURE: mydcv1 COUNT: 1 6) STEP: 2017 OPEN: 2016 CLOSE: 2017 MEASURE: mydcv1 COUNT: 1 7) OPEN: 2017</pre>
Example	<p>To display the existing scan list:</p> <pre>print(scan.list())</pre>

scan.measurecount	
Attribute	Set or query the measure count value for scanning.
Usage	<p>To read the count:</p> <pre>count = scan.measurecount</pre> <p>count: Present measure count value being used</p> <p>To write the count:</p> <pre>scan.measurecount = count</pre> <p>count: Value to set the measure count. Valid range: 1 to 450000.</p>
Remarks	<p>This attribute sets the measure count in the trigger model. During a scan, the Series 3700 will iterate through the sequence event detector and measure action of the trigger model this many times. After performing this count iterations, the Series 3700 will return to check the scan count.</p> <p>The reset value for this attribute is 1.</p>
Details	<p>The measure count value:</p> <ul style="list-style-type: none"> • is the number of measurements to take per scan step • must be set before a scan is started • applies to all scan steps in the list (the ones already existing in the list as well as the new ones to be added before the scan is started)
Example	<p>To set the measure count to 5:</p> <pre>scan.measurecount = 5</pre>

scan.mode	
Attribute	Use to set or query the scan mode value.
Usage	<p>To read the scan mode value:</p> <pre>init = scan.mode</pre> <p>init: The present scan mode setting</p> <p>To write the scan mode value:</p> <pre>scan.mode = init</pre> <p>init: The desired value of set the scan mode setting. Use one of the following:</p> <ul style="list-style-type: none"> • scan.MODE_OPEN_ALL or 0: indicates if an <code>openall</code> on all slots should be performed before a scan starts (default setting). • scan.MODE_OPEN_SELECTIVE or 1: indicates that an intelligent open takes place (see remarks) • scan.MODE_FIXED_ABR or 2: this setting is equivalent to MODE_OPEN_SELECTIVE plus the following: <ul style="list-style-type: none"> -close all required backplane relays before start of scan -does not open or close these backplane relays during the scan -does not open these backplane relays at the end of scan

scan.mode	
Remarks	<p>This attribute, when set to scan.MODE_OPEN_ALL, indicates an <code>openall</code> on all slots should be performed before a scan starts. Otherwise, when scan.MODE_OPEN_SELECTIVE, an intelligent open takes place as follows:</p> <p>If all steps being scanned have a function value of "nofunction" with their DMM configuration then:</p> <ul style="list-style-type: none"> • Open all channels and backplane relays involved in scanning. • If a closed channel or backplane relay is not involved in scanning it will remain closed during the scan. <p>If any step has a DMM configuration with a function set to something other than "nofunction" then:</p> <ul style="list-style-type: none"> • Open analog backplane relays 1 and 2 on all slots. • Open any common side ohms backplane relays on all slots. • Open any amps channels on all slots. • Open all channels and backplane relays involved in scanning. • If a closed channel or backplane relay is not involved in scanning it will remain closed during the scan. • Open all channels on any bank when any backplane relay on that bank is involved in scanning. <p>The reset value as well as the default setting for this attribute is scan.MODE_OPEN_ALL.</p>
Also see	scan.reset() (on page 13-242)
Example	To set the scan mode setting for scan to open selective: <code>scan.mode = scan.MODE_OPEN_SELECTIVE</code>

scan.nobufferbackground()	
Function	Specifies that no reading buffer is used during scanning. Scan is run in background mode.
Usage	<pre>state, scancount, stepcount = scan.nobufferbackground()</pre> <p>state: The result of scanning:</p> <ul style="list-style-type: none"> • scan.EMPTY or 0 • scan.BUILDING or 1 • scan.RUNNING or 2 • scan.ABORTED or 3 • scan.FAILED or 4 • scan.FAILED_INIT or 5, • scan.SUCCESS or 6 <p>scancount: is current scan count completed</p> <p>stepcount: is current step count completed</p>

scan.nobufferbackground()	
Remarks	<p>This command executes the scan in the background. No reading buffer will be used and an error will be generated if one is specified. This command starts the scan. Prior to using this command use scan.create() (on page 13-234) and scan.add() (on page 13-230) to setup scan elements.</p> <p>Because scan is running in the background, use scan.state (on page 13-243) function to see current status of scanning.</p> <p>This functions return parameters do not include the DMM reading.</p>
Also see	<p>scan.add() (on page 13-230)</p> <p>scan.background() (on page 13-232) (see this command to run a background scan with a reading buffer)</p> <p>scan.create() (on page 13-234)</p> <p>scan.execute() (on page 13-236)</p> <p>scan.list() (on page 13-237)</p> <p>scan.nobufferexecute() (on page 13-241)</p> <p>scan.state() (on page 13-243)</p>
Example	<p>To run the scan in the background without using a reading buffer.</p> <pre>scan.nobufferbackground()</pre>

scan.nobufferexecute()	
Function	Specifies that no reading buffer is used during scanning. Scan is run in immediate mode.
Usage	<pre>state, scancount, stepcount = scan.nobufferexecute()</pre> <p>state: The result of scanning. Use one of the following:</p> <ul style="list-style-type: none"> • scan.EMPTY or 0 • scan.BUILDING or 1 • scan.RUNNING or 2 • scan.ABORTED or 3 • scan.FAILED or 4 • scan.FAILED_INIT or 5, • scan.SUCCESS or 6 • scancount: the current scan count completed • stepcount: the current step count completed
Remarks	<p>This command specifies the reading buffer is not used during scanning and runs the scan in immediate mode. This command starts the scan. Prior to using this command use scan.create() (on page 13-234) and scan.add() (on page 13-230) to setup scan elements. An error will be generated if a reading buffer is specified.</p> <p>The command will not exit execution until scanning completes or aborted by user.</p> <p>This functions return parameters do not include the DMM reading.</p>

scan.nobufferexecute()	
Also see	scan.add() (on page 13-230) scan.background() (on page 13-232) scan.create() (on page 13-234) scan.execute() (on page 13-236) (see this command to run a scan with a reading buffer) scan.list() (on page 13-237) scan.nobufferbackground() (on page 13-240) scan.state() (on page 13-243)
Example	To start a scan in immediate mode without using a reading buffer: <pre>scan.nobufferexecute()</pre>
scan.reset()	
Function	Resets the scanning aspects of the system to factory default settings.
Usage	<code>scan.reset()</code>
Remarks	<p>This command will only reset the scan aspects of the system to factory default settings. Settings affected are:</p> <ul style="list-style-type: none"> • Trigger model settings within the scan logical device get reset to factory default settings. Settings affected are those controlled by scan.bypass (on page 13-233), scan.measurecount (on page 13-238), scan.mode() (on page 13-239), and scan.scancount (on page 13-242). Also affected are the stimulus settings for arming a scan (scan.trigger.arm.stimulus (on page 13-244)), channel stepping (scan.trigger.channel.stimulus (on page 13-246)), and measuring (scan.trigger.measure.stimulus (on page 13-247) and scan.trigger.sequence.stimulus (on page 13-249)). • The scan list gets cleared so that if the scan list is queried after a reset, the response will be an empty scan. <p>The rest of the settings are unaffected. To reset, the entire system to factory default settings use the <code>reset</code> command.</p>
Also see	channel.reset() (on page 13-68) dmm.reset() (on page 13-161) reset() (on page 13-230)
Example	To perform a reset on the scan aspects of the system: <pre>scan.reset()</pre>

scan.scancount	
Attribute	Set or query the scan count value.
Usage	<p>To read the count:</p> <pre>count = scan.scancount</pre> <p>count: Present scan count value being used</p> <p>To write the count:</p> <pre>scan.scancount = count</pre> <p>count: Value to set the scan count. Valid range: 1 to 32000.</p>
Remarks	<p>This attribute sets the scan count in the trigger model. During a scan, the Series 3700 will iterate through the arm layer of the trigger model this many times. After performing this count iterations, the Series 3700 will return to idle.</p> <p>The reset value for this attribute is 1.</p>
Example	<p>To set the arm count to 5:</p> <pre>scan.scancount = 5</pre>

scan.state()	
Function	Command to use when running a scan in the background to see present state.
Usage	<code>ScanState = scan.state()</code>
Details	Command to use when running a scan in the background to see present state. See scan.execute() (on page 13-236) or scan.background() (on page 13-232) for details on output. This output matches the return values of execute or background command.
Also see	scan.background() (on page 13-232) scan.execute() (on page 13-236)
Example	<p>To see the current scan state:</p> <pre>ScanState = scan.state() print(ScanState)</pre> <p>or</p> <pre>print(scan.state())</pre>

scan.stepcount	
Attribute	Attribute to query to see number of steps in the present scan.
Usage	To read the number of steps in the present scan: <code>ScanStepCount = scan.stepcount</code>
Remarks	This is a read only attribute. It is set by the number of steps in the present scan when the scan was created and/or steps added.
Also see	scan.add() (on page 13-230)
Example	To see the current scan state: <code>print(scan.stepcount)</code>

scan.trigger.arm.clear()	
Function	Clear the arm event detector.
Usage	<code>scan.trigger.arm.clear()</code>
Remarks	This function will set the arm event detector of the trigger model to the undetected state.
Example	To clear the arm event detector: <code>scan.trigger.arm.clear()</code>

scan.trigger.arm.set()	
Function	Set the arm event detector to the detected state.
Usage	<code>scan.trigger.arm.set()</code>
Remarks	This function will set the arm event detector of the trigger model to the detected state.
Example	To set the arm event detector to the detected state: <code>scan.trigger.arm.set()</code>

scan.trigger.arm.stimulus	
Attribute	Arm event detector trigger selection.
Usage	To read the trigger stimulus: <code>eventid = scan.trigger.arm.stimulus</code> eventid: Present trigger stimulus being used for the arm layer. To write the trigger stimulus: <code>scan.trigger.arm.stimulus = eventid</code> eventid: Stimulus source to set for the arm layer

scan.trigger.arm.stimulus	
Remarks	<p>This attribute selects which event(s) will cause the arm event detector to enter the detected state. Set this attribute to 0 to bypass waiting for an event.</p> <p>eventid may be one of the following (existing trigger event IDs):</p> <ul style="list-style-type: none"> • digio.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line. • display.trigger.EVENT_ID: The trigger key on the front panel is pressed. • trigger.EVENT_ID: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event. • trigger.blender[N].EVENT_ID: A combination of configured events has occurred. • trigger.timer[N].EVENT_ID: A delay expired. • tsplink.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line. • lan.trigger[N].EVENT_ID • scan.trigger.EVENT_SCAN_READY: Scan Ready Event. • scan.trigger.EVENT_SCAN_START: Scan Start Event • scan.trigger.EVENT_CHANNEL_READY: Channel Ready Event • scan.trigger.EVENT_MEASURE_COMP: Measure Complete Event • scan.trigger.EVENT_SEQUENCE_COMP: Sequence Complete Event • scan.trigger.EVENT_SCAN_COMP: Scan Complete Event • scan.trigger.EVENT_IDLE: Idle Event <hr/> <p>NOTE Use the ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because it may need to change when enhancements are added to the instrument.</p>
Example	<p>To set trigger stimulus of the arm event detector to line 3 of digital I/O:</p> <pre>scan.trigger.arm.stimulus = digio.trigger[3].EVENT_ID</pre> <p>To clear trigger stimulus of the arm event detector:</p> <pre>scan.trigger.arm.stimulus = 0</pre>

scan.trigger.channel.clear()	
Function	Clear the channel event detector.
Usage	<code>scan.trigger.channel.clear()</code>
Remarks	This function will clear the channel event detector of the trigger model to the undetected state.
Example	<p>To clear the channel event detector:</p> <pre>scan.trigger.channel.clear()</pre>

scan.trigger.channel.set()	
Function	Set the channel event detector to the detected state.
Usage	<code>scan.trigger.channel.set()</code>
Remarks	This function will set the channel event detector of the trigger model to the detected state.
Example	<code>scan.trigger.channel.set()</code>
scan.trigger.channel.stimulus	
Attribute	Channel event detector stimulus selection.
Usage	<p>To read the trigger stimulus:</p> <pre>eventid = scan.trigger.channel.stimulus</pre> <p>eventid: Present trigger stimulus being used for the channel action.</p> <p>To write the stimulus:</p> <pre>scan.trigger.channel.stimulus = eventid</pre> <p>eventid: Trigger stimulus to set for the channel action</p>
Remarks	<p>This attribute selects which event(s) will cause the channel event detector to enter the detected state. Set this attribute to 0 to reset this event back to its factory default.</p> <p>eventid may be one of the following (existing trigger event IDs):</p> <ul style="list-style-type: none"> • <code>digio.trigger[N].EVENT_ID</code>: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line. • <code>display.trigger.EVENT_ID</code>: The trigger key on the front panel is pressed. • <code>trigger.EVENT_ID</code>: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event. • <code>trigger.blender[N].EVENT_ID</code>: A combination of events has occurred. • <code>trigger.timer[N].EVENT_ID</code>: A delay expired. • <code>tsplink.trigger[N].EVENT_ID</code>: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line. • <code>lan.trigger[N].EVENT_ID</code> • <code>scan.trigger.EVENT_SCAN_READY</code>: Scan Ready Event. • <code>scan.trigger.EVENT_SCAN_START</code>: Scan Start Event • <code>scan.trigger.EVENT_CHANNEL_READY</code>: Channel Ready Event • <code>scan.trigger.EVENT_MEASURE_COMP</code>: Measure Complete Event • <code>scan.trigger.EVENT_SEQUENCE_COMP</code>: Sequence Complete Event • <code>scan.trigger.EVENT_SCAN_COMP</code>: Scan Complete Event • <code>scan.trigger.EVENT_IDLE</code>: Idle Event <p>NOTE Use the ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because it may need to change when enhancements are added to the instrument</p>

scan.trigger.channel.stimulus	
Example	To set trigger stimulus of the channel event detector to scan start event: <pre>scan.trigger.channel.stimulus = scan.trigger.EVENT_SCAN_START</pre>
scan.trigger.clear()	
Function	Clear the trigger model.
Usage	<pre>scan.trigger.clear()</pre>
Remarks	This function will set the channel, measure and sequence event detectors of the trigger model to the undetected state.
Example	To clear the trigger model: <pre>scan.trigger.clear()</pre>
scan.trigger.measure.clear()	
Function	Clear the measure event detector.
Usage	<pre>scan.trigger.measure.clear()</pre>
Remarks	This function will set the measure event detector of the trigger model to the undetected state.
Example	To clear the measure event detector: <pre>scan.trigger.measure.clear()</pre>
scan.trigger.measure.set()	
Function	Set the measure event detector to the detected state.
Usage	<pre>scan.trigger.measure.set()</pre>
Remarks	This function will set the measure event detector of the trigger model to the detected state.
Example	To set the measure event detector to the detected state: <pre>scan.trigger.measure.set()</pre>
scan.trigger.measure.stimulus	
Attribute	Measure event detector trigger stimulus selection.
Usage	To read the trigger stimulus: <pre>eventid = scan.trigger.measure.stimulus</pre> eventid: the present trigger stimulus being used for the measure event To write the trigger stimulus: <pre>scan.trigger.measure.stimulus = eventid</pre> eventid: the trigger stimulus to set for the measure event

scan.trigger.measure.stimulus	
Remarks	<p>This attribute selects which event(s) will cause the measure event detector to enter the detected state. Set this attribute to 0 to bypass waiting for an event.</p> <p>eventid may be one of the following (existing trigger event IDs):</p> <ul style="list-style-type: none"> • digio.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line. • display.trigger.EVENT_ID: The trigger key on the front panel is pressed. • trigger.EVENT_ID: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event. • trigger.blender[N].EVENT_ID: A combination of events has occurred. • trigger.timer[N].EVENT_ID: A delay expired. • tsplink.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line. • lan.trigger[N].EVENT_ID • scan.trigger.EVENT_SCAN_READY: Scan Ready Event. • scan.trigger.EVENT_SCAN_START: Scan Start Event • scan.trigger.EVENT_CHANNEL_READY: Channel Ready Event • scan.trigger.EVENT_MEASURE_COMP: Measure Complete Event • scan.trigger.EVENT_SEQUENCE_COMP: Sequence Complete Event • scan.trigger.EVENT_SCAN_COMP: Scan Complete Event • scan.trigger.EVENT_IDLE: Idle Event <p>Use this to start a set of measure count readings that are being triggered by a single event. To pace each reading by an event, use the sequence trigger stimulus.</p> <hr/> <p>NOTE Use the ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because it may need to change when enhancements are added to the instrument.</p>
Also see	scan.trigger.sequence.stimulus (on page 13-249)
Example	<p>To set trigger stimulus of the measure event detector to channel ready event:</p> <pre>scan.trigger.measure.stimulus = scan.trigger.EVENT_CHAN_READY</pre>

scan.trigger.sequence.clear()	
Function	Clear the sequence event detector.
Usage	<code>scan.trigger.sequence.clear()</code>
Remarks	This function will set the sequence event detector to the undetected state.
Example	<p>To clear the sequence event detector:</p> <pre>scan.trigger.sequence.clear()</pre>

scan.trigger.sequence.set()	
Function	Set the sequence event detector to the detected state.

scan.trigger.sequence.set()	
Usage	<code>scan.trigger.sequence.set()</code>
Remarks	This function will set the sequence event detector to the detected state.
Example	To set the sequence event detector to the detected state: <code>scan.trigger.sequence.set()</code>

scan.trigger.sequence.stimulus	
Attribute	Sequence event detector trigger stimulus selection.
Usage	To read the trigger stimulus: <code>eventid = scan.trigger.sequence.stimulus</code> eventid : Present trigger source being used for the sample event. To write the trigger stimulus: <code>scan.trigger.sequence.stimulus = eventid</code> eventid : Trigger source to set for the sample event
Remarks	This attribute selects which event(s) will cause the sequence event detector to enter the detected state. Set this attribute to 0 to bypass waiting for an event. eventid may be one of the following (existing trigger event IDs): <ul style="list-style-type: none"> • <code>digio.trigger[N].EVENT_ID</code>: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line. • <code>display.trigger.EVENT_ID</code>: The trigger key on the front panel is pressed. • <code>trigger.EVENT_ID</code>: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event. • <code>trigger.blender[N].EVENT_ID</code>: A combination of events has occurred. • <code>trigger.timer[N].EVENT_ID</code>: A delay expired. • <code>tsplink.trigger[N].EVENT_ID</code>: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line. • <code>lan.trigger[N].EVENT_ID</code> • <code>scan.trigger.EVENT_SCAN_READY</code>: Scan Ready Event. • <code>scan.trigger.EVENT_SCAN_START</code>: Scan Start Event • <code>scan.trigger.EVENT_CHANNEL_READY</code>: Channel Ready Event • <code>scan.trigger.EVENT_MEASURE_COMP</code>: Measure Complete Event • <code>scan.trigger.EVENT_SEQUENCE_COMP</code>: Sequence Complete Event • <code>scan.trigger.EVENT_SCAN_COMP</code>: Scan Complete Event • <code>scan.trigger.EVENT_IDLE</code>: Idle Event Use this to pace each one of the measure count readings with an event. If you don't want to pace the reading, then set this stimulus to 0. NOTE Use the ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because they may need to change when enhancements are added to the instrument.

scan.trigger.sequence.stimulus	
Also see	scan.trigger.measure.stimulus (on page 13-247)
Example	To set trigger stimulus of the sequence event detector to channel ready event: <pre>scan.trigger.sequence.stimulus = scan.trigger.EVENT_CHAN_READY</pre>

schedule functions and attributes

Use these functions to configure the scheduled alarm events. These events are generated at times defined by these ICLs. The generated triggers can then be used to drive other event driven components, such as digital I/O, scan trigger model, event blenders, etc. If the system time is driven by IEEE-1588, then the alarms use that time clock. Otherwise, the alarms use the standard CPU clock.

schedule.alarm[x].enable	
Attribute	Enable or disable an alarm.
Usage	<pre>status = schedule.alarm[x].enable</pre> status: The enable or disable status of the alarm. Use one of the following: <ul style="list-style-type: none"> • <code>eventlog.ENABLE</code> or 1: enable alarm • <code>eventlog.DISABLE</code> or 0: disable alarm
Remarks	When enabling an alarm with a set start time in the past, the alarm executes immediately. When used to start a scan, an alarm time in the past may be missed by the scan start. The scan clears any pending triggers before it begins and therefore will miss the any trigger generated from the alarm enable. To ensure this does not happen, start the scan in the background then enable the alarm.
Example	Enable an alarm: <pre>schedule.alarm[1].enable = 1</pre>

schedule.alarm[x].EVENT_ID	
Function	The event identifier constant for use with the stimulus attribute.
Usage	<pre>schedule.alarm[x].EVENT_ID</pre>
Example	Command a scan to occur when alarm 1 fires: <pre>scan.trigger.arm.stimulus = schedule.alarm[1].EVENT_ID</pre>

schedule.alarm[x].fractionalseconds	
Attribute	The fractional seconds portion of the alarm time.
Usage	<pre>schedule.alarm[x].fractionalseconds [= fraction]</pre>
Remarks	1588 has too much resolution to represent in a single floating point value so the alarm times are split into two values (seconds and fractional seconds).

schedule.alarm[x].fractionalseconds	
Example	Create an alarm to occur 60.25 seconds from current time in UTC seconds <pre>sec,ns = os.time() schedule.alarm[1].seconds = sec + 60 schedule.alarm[1].fractionalseconds = ns + 0.5</pre>
schedule.alarm[x].period	
Attribute	The time, in seconds, between adjacent firings of the alarm.
Usage	<code>schedule.alarm[x].period</code>
Example	Set period of 0.5 seconds between firings of alarms after initial alarm <pre>schedule.alarm[1].period = 0.5</pre>
schedule.alarm[x].ptpseconds	
Attribute	The seconds portion of the alarm time in PTP seconds (see <code>ptp.utcoffset</code>).
Usage	<code>schedule.alarm[x].ptpseconds [= seconds]</code>
Remarks	1588 has too much resolution to represent in a single floating point value so the alarm times are split into two values (seconds and fractional seconds).
Example	Create an alarm to occur 30 seconds from current time in PTP seconds. <pre>sec,ns = ptp.time() schedule.alarm[1].ptpseconds = sec + 30</pre>
schedule.alarm[x].repetition	
Attribute	Specifies the number of times an alarm will repeat after the first alarm firing (the alarm will fire a total of <code>count+1</code> times). If 0 and period is non-zero, the alarm will fire "forever". Once an alarm begins, the repetition will count down for each trigger generated. It will end at zero (0). You must set this repetition back to some value if you intend to reissue the alarm again. Otherwise, the alarm will either not fire (if period is zero) or will fire "forever" (if period is non-zero).
Usage	<code>schedule.alarm[x].repetition [= count]</code>
Example	Set alarm to fire 10 times <pre>schedule.alarm[1].repetition = 10</pre>
schedule.alarm[x].seconds	
Attribute	The seconds portion of the alarm time in UTC seconds (see <code>ptp.utcoffset</code>).
Usage	<code>schedule.alarm[x].seconds [= seconds]</code>
Remarks	1588 has too much resolution to represent in a single floating point value so the alarm times are split into two values (seconds and <code>fractionalseconds</code>).

schedule.alarm[x].seconds	
Example	<p>Create an alarm to occur on March 15, 2008 at 10AM in UTC seconds:</p> <pre>local l_myTime l_myTime = os.time{year = 2008, month = 3, day = 15, hour = 10} schedule.alarm[1].seconds = l_myTime</pre>

schedule.disable()	
Function	Disable all alarms.
Usage	<code>schedule.disable()</code>
Example	<code>schedule.disable()</code>

setup functions and attributes

Use the functions and attribute in this group to save/recall setups and to set the power-on setup.

setup.cards()	
Function	Queries the card model number for each slot of a saved setup.
Usage	<p><code>CardModels = setup.cards()</code> -- for cards associated with internally saved setup</p> <p><code>CardModels = setup.cards('/usb1/<filename>.set')</code> -- for cards associated with the <filename> on thumb drive. Note: The .set extension needs to be on the filename specified.</p> <p>CardModels: A comma-delimited string listing card model for each slot.</p>
Remarks	This function will return a comma-delimited string listing the card model for each slot in the system (from 1 to 6) for the desired saved setup. If no card was installed in the slot when the setup was saved, a 0 is returned as the card model number.
Details	<p>To successfully recall a setup, the card models need to match or a card needs to be installed in an empty slot of the setup configuration. Otherwise, an error will be generated.</p> <p>To recall a setup that was saved when the specified slot card is unavailable, assign a pseudo card to that slot. For example, assume Slot 3 contained a Model 3720 when saved. To recall that setup without an actual Model 3720 installed in the system, assign a pseudo 3720 to Slot 3 with:</p> <pre>slot[3].pseudocard = slot.PSEUDO_3720</pre> <p>When recalling a setup, the slot needs to contain the necessary model, but it doesn't matter if that model is present with a real card or pseudo card. The instrument does not differentiate between the two when recalling a setup; they are treated the same. In this example, both would be seen as the system having a Model 3720 in Slot 3.</p>

setup.cards()	
Example	<p>To query the cards of the internal saved setup:</p> <pre>CardModels = setup.cards() print(CardModels)</pre> <p>Output: 3722, 0, 0, 0, 0, 0</p> <p>To query the cards associated with mysetup.set on thumb drive:</p> <pre>print(setup.card('/usb1/mysetup.set')) --> 0, 3723, 3722, 3720, 0, 0</pre>
setup.poweron	
Attribute	The setup to recall when the unit is turned on.
Usage	<p>To read the power-on setup:</p> <pre>n = setup.poweron</pre> <p>To write the power-on setup:</p> <pre>setup.poweron = n</pre> <p>n: Setup number to recall on power up (0 or 1).</p>
Remarks	Setting this attribute to 0 causes the unit to power up to the factory default (reset) setup. A setting of 1 causes the unit to power up using a user setup that was previously saved internally.
Example	<p>Sets unit to power on to the factory default settings:</p> <pre>setup.poweron = 0</pre>
setup.recall()	
Function	Recalls settings from a saved setup.
Usage	<pre>setup.recall(location)</pre> <p>location: Setup number to recall (0, 1, or "/usb1/<filename>").</p> <p>0: Reset setup.</p> <p>1: Internal setup.</p> <p><filename>: Use the name of the desired file contained on a USB flash drive.</p>
Remarks	<p>If a number is sent as the parameter:</p> <ul style="list-style-type: none"> The number is interpreted as a setup number and the setup is recalled from internal memory. Setting this attribute to 0 recalls the factory default (reset) setup. Setting this attribute to 1 recalls the user saved setup from internal memory. <p>If a string is sent as the parameter:</p> <ul style="list-style-type: none"> The string is interpreted as a path and filename and the setup is recalled from the corresponding file on the USB flash drive. The path may be absolute or relative to the current working directory.

setup.recall()	
Example	<p>To recall factory default settings: <code>setup.recall(0)</code></p> <p>To recall the user-setup (internal): <code>setup.recall(1)</code></p> <p>To recall a user saved setup stored in a file named KEITHLEY_3730 on a USB flash drive: <code>setup.recall("/usb1/KEITHLEY_3730.set")</code></p>
setup.save()	
Function	Saves the present setup as a user-setup.
Usage	<p>To save to the internal memory location, send no parameters with function: <code>setup.save()</code></p> <p>To save to the USB flash drive: <code>setup.save(location)</code></p> <p>location: Setup location to save. Use the format <code>"/usb1/<filename>"</code> where <filename> is the name of the desired file contained on a USB flash drive.</p>
Remarks	<p>This function overwrites any previous values with the present setup. When saving a setup to an attached USB flash drive, specify <code>"/usb1/"</code> at the start of the filename. The <code>.set</code> is appended to the filename. Any specified file extension other than <code>.set</code> will generate errors.</p> <p>Valid destination filename examples:</p> <pre>setup.save('/usb1/mysetup')</pre> <pre>setup.save('/usb1/mysetup.set')</pre> <p>Invalid destination filename examples:</p> <pre>setup.save('/usb1/mysetup.stp')</pre> <p>-Invalid extension due to ending period followed by no letters for extension. <pre>setup.save('/usb1/mysetup.txt')</pre></p> <p>-Invalid extension. Use <code>.set</code> or do not specify (no period) <pre>setup.save('/usb1/mysetup.txt.set')</pre></p> <p>-invalid extension because 2 periods specified (<code>mysetup_txt.set</code> would be correct).</p> <hr/> <p>NOTE The setup files saved to the USB flash drive will always have an extension of <code>.set</code>.</p>
Example	<p>To save the present setup as the internal user setup: <code>setup.save()</code></p> <p>To save a setup to a file named KEITHLEY_3730 on a USB flash drive: <code>setup.save("/usb1/KEITHLEY_3730")</code></p>

slot[X] attributes

The attributes in this group indicate whether a card in slot X (where X = 1 to 6) supports different features such as pole settings, voltage or 2-wire measurements, etc.,. To query an attribute, use the print command sending the attribute as an argument. For example:

```
print(slot[1].idn)
```

will output a comma separated string that contains the model number, description, firmware revision and serial number of the card installed in Slot 1.

slot[X].commonsideohms	
Attribute	Indicates whether a card in slot X supports common-side 4-wire ohm channels.
Usage	<code>commonsideohms = slot[X].commonsideohms</code> [X]: Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support common-side 4-wire ohm channels. In these cases, the return value will be nil. If common side 4-wire ohm channels are supported, the returned value will be 1.
Example	To query if Slot 1 supports common-side 4-wire ohm channels: <code>CommonSideOhms1 = slot[1].commonsideohms</code>

slot[X].digio	
Attribute	Indicates whether a card in slot X supports digital I/O channels or not.
Usage	<code>digio = slot[X].digio</code> [X]: Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support digital I/O channels. In these cases, the return value will be nil. If digital I/O channels are supported, the returned value will be 1.
Example	To query if Slot 1 supports digio channels: <code>Digio1 = slot[1].digio</code>

slot[X].endchannel.amps	
Attribute	The ending channel that supports amps measurements.
Usage	<code>end = slot[X].endchannel.amps</code> [X]: Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support amps channels. In these cases, the return value will be nil. If supported, the return value will be a number representing the ending channel. If only one channel supports amps, the ending channel will match the starting channel number.

slot[X].endchannel.amps	
Example	To query for ending amps channel on Slot 4: <code>EndAmpsChan = slot[4].endchannel.amps</code>

slot[X].endchannel.analogoutput	
Attribute	The ending channel that supports a digital analog output (DAC).
Usage	<code>end = slot[X].endchannel.analogoutput</code> [X] : Slot number (1 to 6)
Remarks	This attribute does not exist for a slot if a card is not installed or the card installed does not support analog output channels. In these cases, the return value is <code>nil</code> . If supported, the return value is a number representing the ending channel. If only one channel supports analog output, the ending channel matches the starting channel number.
Example	Query for ending analog output channel on Slot 4: <code>EndAnalogoutputChan = slot[4].endchannel.analogoutput</code>

slot[X].endchannel.digitalio	
Attribute	The ending channel that supports digital input and output.
Usage	<code>end = slot[X].endchannel.digitalio</code> [X] : Slot number (1 to 6)
Remarks	This attribute does not exist for a slot if a card is not installed or the card installed does not support digital I/O channels. In these cases, the return value is <code>nil</code> . If supported, the return value is a number representing the ending channel. If only one channel supports digital I/O, the ending channel matches the starting channel number.
Example	To query for ending digital input channel on Slot 4: <code>EndDigitalIOChan = slot[4].endchannel.digitalio</code>

slot[X].endchannel.isolated	
Attribute	The ending channel that supports isolated.
Usage	<code>end = slot[X].endchannel.isolated</code> [X] : Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support isolated channels. In these cases, the return value will be <code>nil</code> . If supported, the return value will be a number representing the ending channel. If only one channel supports isolated, the ending channel will match the starting channel number.
Example	Query for ending isolated channel on Slot 4: <code>EndIsolatedChan = slot[4].endchannel.isolated</code>

slot[X].endchannel.totalizer	
Attribute	The ending channel that supports a totalizer.
Usage	<code>end = slot[X].endchannel.totalizer</code> [X]: Slot number (1 to 6)
Remarks	This attribute does not exist for a slot if a card is not installed or the card installed does not support totalizer channels. In these cases, the return value is <code>nil</code> . If supported, the return value is a number representing the ending channel. If only one channel supports totalizer, the ending channel matches the starting channel number.
Example	To query for starting totalizer channel on Slot 4: <code>EndTotalizerChan = slot[4].endchannel.totalizer</code>

slot[X].endchannel.voltage	
Attribute	The ending channel that supports voltage or 2-wire measurements.
Usage	<code>end = slot[X].endchannel.voltage</code> [X]: Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support voltage channels. In these cases, the return value will be <code>nil</code> . If supported, the return value will be a number representing the ending channel. If only one channel supports voltage, the ending channel will match the starting channel number.
Example	To query for ending voltage channel on Slot 4: <code>EndVoltageChan = slot[4].endchannel.voltage</code>

slot[X].idn	
Attribute	Returns a string containing model number, description, firmware revision and serial number of the card in slot X.
Usage	<code>card_idn = slot[X].idn</code> [X]: Slot number (1 to 6)
Remarks	Returns a comma separated string that contains the model number, description, firmware revision and serial number of the card installed in slot X. This attribute will return a string indicating an empty slot if an actual card is not installed and if the slot is not configured for a pseudo card. For pseudocards, the response will be model number, description and firmware revision. When queried, the return value will have "Pseudo" before the card description. For example, if a Model 3720 Pseudocard is installed in Slot 3: <code>print(slot[3].idn) → 3720,Pseudo Dual 1x30 Multiplexer,00.00a</code>
Example	To query for idn information for the card in Slot 1: <code>card1_idn = slot[1].idn</code>

slot[X].interlock.override	
Attribute	Indicates if a card should error on closing backplane relays if interlock is disengaged.
Usage	<p>To read interlock override setting: <code>value = slot[X].interlock.override</code></p> <p>[X]: Slot number (1 to 6)</p> <p>To write interlock override setting: <code>slot[X].interlock.override = value</code></p> <p>[X]: Slot number (1 to 6)</p> <p>value: Represents the desired state of the interlock override. Set to one of the following:</p> <ul style="list-style-type: none"> • slot.ON or 1 • slot.OFF or 0 (default setting and reset value)
Remarks	<p>This attribute exists only for installed cards that support detecting an interlock break. Otherwise, the return value will be nil.</p> <p>If card supports detecting an interlock break, set this attribute to the desired response.</p> <p>To enable interlock override on the card, set to slot.ON. Otherwise (if an override performed on card is not desired), set to slot.OFF. This setting applies to all interlocks on the card.</p>
Example	<p>To not have an override performed after detecting an interlock break on Slot 3: <code>slot[3].interlock.override = slot.OFF</code></p>

slot[X].interlock.state	
Attribute	Indicates the interlock state of a card.
Usage	<p>To read the interlock state: <code>value = slot[X].interlock.state</code></p> <p>[X]: Slot number (1 to 6)</p> <p>value: Represents whether the interlocks are engaged or not. Interpret the interlock state values as follows:</p> <p><code>nil</code>: no card is installed or card installed does not support interlocks</p> <p><code>0</code>: interlocks 1 and 2 are disengaged on card.</p> <p><code>1</code>: interlock 1 is engaged while interlock 2, if it exists, is disengaged.</p> <p><code>2</code>: interlock 1 is disengaged while interlock 2 is engaged.</p> <p><code>3</code>: interlocks 1 and 2 are engaged on card.</p>
Remarks	<p>This attribute will not exist for a slot if a card is not installed or the card installed does not support detecting an interlock break. In these cases, the return value will be nil.</p> <p>Use this attribute to query the interlock state for cards that support detecting interlock break.</p>

slot[X].interlock.state	
Example	To query the interlock state on Slot 3: <code>print(slot[3].interlock.state)</code>

slot[X].isolated	
Attribute	Indicates whether a card in slot X supports isolated channels or not.
Usage	<code>isolated_chans = slot[X].isolated</code> [X] : Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support isolated channels. In these cases, the return value will be nil. If isolated channels are supported, the returned value will be 1.
Example	To query if Slot 1 supports isolated channels: <code>IsolatedChan1 = slot[1].isolated</code>

slot[X].matrix	
Attribute	Indicates whether a card in slot X supports matrix channels or not.
Usage	<code>matrix = slot[X].matrix</code> [X] : Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support matrix channels. In these cases, the return value will be nil. If matrix channels are supported, the returned value will be 1.
Example	To query if Slot 1 supports matrix channels: <code>Matrix1 = slot[1].matrix</code>

slot[X].maxvoltage	
Attribute	Returns the maximum voltage supported by the card in slot X.
Usage	<code>maxvolts = slot[X].maxvoltage</code> [X] : Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not have a maximum voltage setting. In these cases, the return value will be nil. This attribute value represents the maximum voltage of all channels on a particular slot.
Example	To query the maximum voltage for card on Slot 2: <code>maxvolts2 = slot[2].maxvoltage</code>

slot[X].multiplexer	
Attribute	Indicates whether a card in slot X supports multiplexer channels or not.
Usage	<code>mux_chans = slot[X].multiplexer</code> [X] : Slot number (1 to 6)

slot[X].multiplexer	
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support multiplexer channels. In these cases, the return value will be nil. If multiplexer channels are supported, the returned value will be 1.
Example	To query if Slot 1 supports multiplexer channels: <code>MuxChan1 = slot[1].multiplexer</code>

slot[X].poles.four	
Attribute	Indicates if the card supports four-pole.
Usage	<code>fourpole = slot[X].poles.four</code> [X] : Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support four-pole. In these cases, the return value will be nil. If supported, the return value will be 1.
Example	To query if Slot 3 supports four-pole: <code>FourPole3 = slot[3].poles.four</code>

slot[X].poles.one	
Attribute	Indicates if the card supports one-pole.
Usage	<code>onepole = slot[X].poles.one</code> [X] : Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support one-pole. In these cases, the return value will be nil. If supported, the return value will be 1.
Example	To query if Slot 3 supports one-pole: <code>OnePole3 = slot[3].poles.one</code>

slot[X].poles.two	
Attribute	Indicates if the card supports two-pole.
Usage	<code>twopole = slot[X].poles.two</code> [X] : Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support two-pole. In these cases, the return value will be nil. If supported, the return value will be 1.
Example	To query if Slot 3 supports two-pole: <code>TwoPole3 = slot[3].poles.two</code>

slot[X].pseudocard	
Attribute	Specifies the corresponding pseudo card to implement for the designated slot.
Usage	<p>To read pseudo card for slot: <code>pseudocard = slot[X].pseudocard</code> [X]: Slot number (1 to 6)</p> <p>To write pseudo card for slot: <code>slot[X].pseudocard = pseudocard</code> [X]: Slot number (1 to 6)</p> <p>Set pseudocard to one of the following values</p> <ul style="list-style-type: none"> • <code>slot.PSEUDO_NONE</code> or 0 for no pseudocard selection • <code>slot.PSEUDO_3720</code> or 3720 for Model 3720 Dual 1x30 Multiplexer card simulation • <code>slot.PSEUDO_3721</code> or 3721 for Model 3721 Dual 1x20 Multiplex card simulation • <code>slot.PSEUDO_3722</code> or 3722 for Model 3722 Dual 1x48 Multiplexer card simulation • <code>slot.PSEUDO_3723</code> or 3723 for Model 3723 Dual 1x30 Reed Multiplexer card simulation • <code>slot.PSEUDO_3724</code> or 3724 for Model 3724 Dual 1x30 FET Multiplexer card simulation • <code>slot.PSEUDO_3730</code> or 3730 for Model 3730 6 x 16 High Density Matrix card simulation • <code>slot.PSEUDO_3740</code> or 3740 for Model 3740 32-Channel Isolated Switch card simulation • <code>slot.PSEUDO_3750</code> or 3750 for Model 3750 Multifunction I/O card
Remarks	<p>This attribute only exists for a slot if that slot has no card installed in it. Therefore, trying to use this attribute for a slot with an installed card generates an error when writing and nil response when reading. After assigning a pseudo card, the valid commands and attributes based on that pseudo card now exist for that slot. For example, the <code>slot[X].idn</code> attribute is valid.</p> <p>Changing a slot's pseudocard card definition from a card to <code>none</code> invalidates an existing scan list.</p>
Details	<p>The response to an <code>idn</code> query is model number, description, and firmware revision. When queried, the return value has "Pseudo" before the card description. For example, if a Model 3720 pseudocard is installed in Slot 3:</p> <pre>print(slot[3].idn) → 3720,Pseudo Dual 1x30 Multiplexer,00.00a</pre>
Example	<p>Sets the pseudo card of Slot 6 for Model 3720 card simulation:</p> <pre>slot[6].pseudocard = slot.PSEUDO_3720</pre>

slot[X].startchannel.amps	
Attribute	The starting channel that supports amps measurements.
Usage	<code>start = slot[X].startchannel.amps</code> [X]: Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support amps channels. In these cases, the return value will be nil. If supported, the return value will be a number representing the starting channel.
Example	To query for starting amps channel on Slot 4: <code>StartAmpsChan = slot[4].startchannel.amps</code>

slot[X].startchannel.analogoutput	
Attribute	The starting channel that supports digital analog output (DAC).
Usage	<code>start = slot[X].startchannel.analogoutput</code> [X]: Slot number (1 to 6)
Remarks	This attribute does not exist for a slot if a card is not installed or the card installed does not support analog output channels. In these cases, the return value is nil. If supported, the return value is a number representing the starting channel.
Example	To query for starting analog output channel on Slot 4: <code>StartAnalogOutputChan = slot[4].startchannel.analogoutput</code>

slot[X].startchannel.digitalio	
Attribute	The starting channel that supports digital input and output.
Usage	<code>start = slot[X].startchannel.digitalio</code> [X]: Slot number (1 to 6)
Remarks	This attribute does not exist for a slot if a card is not installed or the card installed does not support digital I/O channels. In these cases, the return value is nil. If supported, the return value is a number representing the starting channel.
Example	To query for starting digital input channel on Slot 4: <code>StartDigitalIOChan = slot[4].startchannel.digitalio</code>

slot[X].startchannel.isolated	
Attribute	The starting channel that supports isolated measurements.
Usage	<code>start = slot[X].startchannel.isolated</code> [X]: Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support isolated channels. In these cases, the return value will be nil. If supported, the return value will be a number representing the starting channel.
Example	To query for starting isolated channel on Slot 4: <code>StartIsolatedChan = slot[4].startchannel.isolated</code>

slot[X].startchannel.totalizer	
Attribute	The starting channel that supports a totalizer.
Usage	<code>start = slot[X].startchannel.totalizer</code> [X]: Slot number (1 to 6)
Remarks	This attribute does not exist for a slot if a card is not installed or the card installed does not support totalizer channels. In these cases, the return value is <code>nil</code> . If supported, the return value is a number representing the starting channel.
Example	To query for starting totalizer channel on Slot 4: <code>StartTotalizerChan = slot[4].startchannel.totalizer</code>

slot[X].startchannel.voltage	
Attribute	The starting channel that supports voltage or 2-wire measurements.
Usage	<code>start = slot[X].startchannel.voltage</code> [X]: Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support voltage channels. In these cases, the return value will be <code>nil</code> . If supported, the return value will be a number representing the starting channel.
Example	To query for starting voltage channel on Slot 4: <code>StartVoltageChan = slot[4].startchannel.voltage</code>

slot[X].tempsensor	
Attribute	Indicates whether a card in slot X supports temperature sensor channels or not.
Usage	<code>temp_sensor = slot[X].tempsensor</code> [X]: Slot number (1 to 6)
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support temperature sensor channels. In these cases, the return value will be <code>nil</code> . If temperature sensor channels are supported, the returned value will be 1.
Example	To query if Slot 1 supports temperature sensor channels: <code>TempSensors1 = slot[1].tempsensors</code>

slot[X].thermal.state	
Attribute	Indicates the thermal state of a card, if supported.
Usage	<code>value = slot[X].thermal.state</code> [X]: Slot number (1 to 6) value: Indicates whether or not the thermal state of card is getting warm to affect the card's specifications. The thermal state's possible return values: <code>nil</code> : no card is installed or card installed does not support thermal state detection. 0: Means the thermal conditions on card are okay. 1: Means thermal conditions of the card are at a point where specs may be affected.

slot[X].thermal.state	
Remarks	This attribute will not exist for a slot if a card is not installed or the card installed does not support thermal state detection. In these cases, the return value will be nil. Use this attribute to query the thermal state only if the card supports detecting thermal state.
Example	To query the thermal state on Slot 3: <code>print(slot[3].thermal.state)</code>

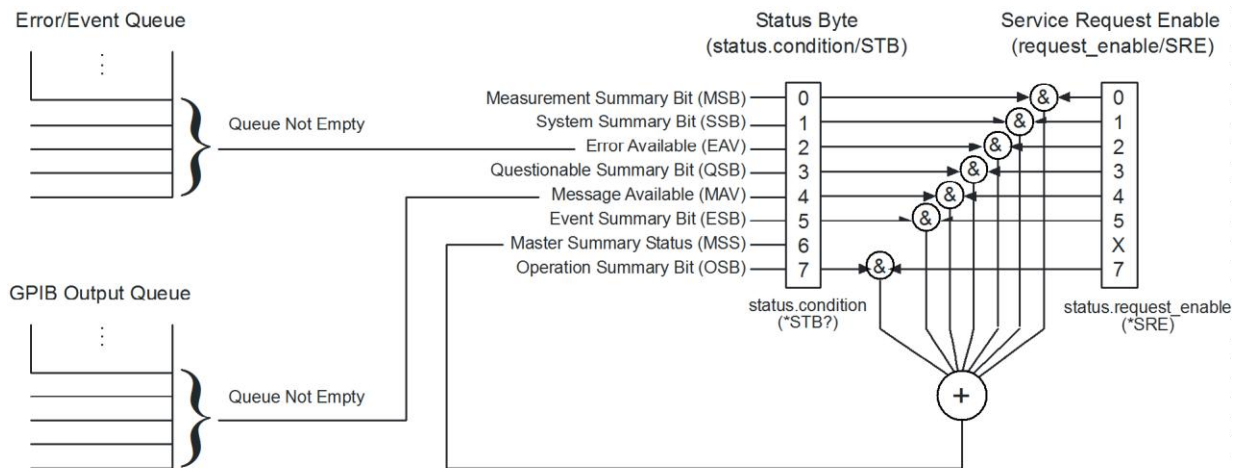
status functions and attributes

The following provides a brief overview of the status model. For details, see [Status Model](#) (on page 12-1).

Status byte and SRQ

The status byte register receives the summary bits of the five status register sets, a master summary bit, 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.

Figure 13-4: Status byte and queues



status.condition	
Attribute	Status byte register.
Usage	Reads the status byte register: <code>statbyte = status.condition</code>
Remarks	<p>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.</p> <p>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.</p> <p>The bits of the status byte register are described as follows:</p> <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> - When using the GPIB serial poll sequence of the Series 3700 to obtain the status byte (serial poll byte), B6 is the RQS bit. - When using <code>status.condition</code> or the <code>*STB?</code> 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.
Example	<p>Reads the status byte:</p> <pre>statbyte = status.condition print(statbyte)</pre> <p>Output: 1.29000e+02</p> <p>The above output indicates that bits B0 (MSS) and B7 (OSB) are set.</p>

status.measurement.*		.condition .enable .event .ntr .ptr
Attribute	Measurement event status register set.	
Usage	<p>To read condition, enable, event, NTR and PTR registers:</p> <pre>measreg = status.measurement.condition measreg = status.measurement.enable measreg = status.measurement.event measreg = status.measurement.ntr measreg = status.measurement.ptr</pre> <p>To write to enable, NTR, and PTR registers:</p> <pre>status.measurement.enable = measreg status.measurement.ntr = measreg status.measurement.ptr = measreg</pre> <p>To set measreg to one of the following values:</p> <ul style="list-style-type: none"> • To clear all bits: 0 • To set ROF bit (B7): status.measurement.READING_OVERFLOW - or - status.measurement.ROF • To set BAV bit (B8): status.measurement.BUFFER_AVAILABLE - or - status.measurement.BAV <p>Other bits are:</p> <ul style="list-style-type: none"> • ULMT1 or UPPER_LIMIT1: Set bit indicates that a reading has exceeded the upper limit 1 value. • LLMT1 or LOWER_LIMIT1: Set bit indicates that a reading has exceeded the lower limit 1 value. • ULMT2 or UPPER_LIMIT2: Set bit indicates that a reading has exceeded the upper limit 2 value. • LLMT2 or LOWER_LIMIT2: Set bit indicates that a reading has exceeded the lower limit 2 value. 	
	<p>measreg can also be set to the decimal weight of the bit to be set. For example:</p> <ul style="list-style-type: none"> • 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 B7 and B8, set measreg to 384 (128 + 256). 	

status.measurement.*		.condition .enable .event .ntr .ptr
Remarks	<p>These attributes are used to read or write to the measurement registers.</p> <p>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.</p> <p>For example, assume value 384 is returned for the register. The binary equivalent is 0000000100000001. This value indicates that bit B7(ROF) and bit B8 (BAV) are set.</p> <p>The used bits of the measurement registers are described as follows:</p> <ul style="list-style-type: none"> • To set bit B0 (LLMT1), set measreg to 1 (2^0). • To set bit B1 (ULMT1): set measreg to 2 (2^1). • To set bit B2 (LLMT2), set measreg to 4 (2^2). • To set bit B3 (ULMT2), set measreg to 8 (2^3). • To set bit B7 (ROF), set measreg to 128 (2^7). • To set bit B8 (BAV), set measreg to 256 (2^8). 	
Example	<p>Sets the BAV bit of the measurement enable register:</p> <pre>status.measurement.enable = status.measurement.BAV</pre>	

status.node_enable	
Attribute	Status node enable register.
	<p>Reads status node enable register: <code>nodeenabreg = status.node_enable</code></p> <p>Writes to system enable register: <code>status.node_enable = nodeenabreg</code></p> <p>Set nodeenabreg to one of the following values:</p> <ul style="list-style-type: none"> • To clear all bits: 0 • To set (enables) MSB bit (B0): <code>status.MEASUREMENT_SUMMARY_BIT</code> - or - <code>status.MSB</code> • To set (enables) EAV bit (B2): <code>status.ERROR_AVAILABLE</code> - or - <code>status.EAV</code> • To set (enables) QSB bit (B3): <code>status.QUESTIONABLE_SUMMARY_BIT</code> - or - <code>status.QSB</code> • To set (enables) MAV bit (B4): <code>status.MESSAGE_AVAILABLE</code> - or - <code>status.MAV</code> • To set (enables) ESB bit (B5): <code>status.EVENT_SUMMARY_BIT</code> - or - <code>status.ESB</code> • To set (enables) MSS bit (B6): <code>status.MASTER_SUMMARY_STATUS</code> - or - <code>status.MSS</code> • To set (enables) OSB bit (B7): <code>status.OPERATION_SUMMARY_BIT</code> - or - <code>status.OSB</code>

status.node_enable	
	<p>nodeenabreg can also be set to the decimal weight of the bit to be set:</p> <p>To set bit B0 (MSB), set nodeenabreg to 1 (2^0).</p> <p>To set bit B2 (EAV), set nodeenabreg to 4 (2^2).</p> <p>To set bit B7 (OSB), set nodeenabreg to 128 (2^7).</p> <p>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).</p>
Remarks	<p>This attribute is used to read or write to the status node enable register.</p> <p>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.</p> <p>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.</p> <p>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.* (on page 13-280) registers).</p> <p>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 (on page 13-264) register.</p>
Example	<p>Sets the MSB bit of the status node enable register:</p> <pre>status.node_enable = status.MSB</pre>

status.node_event	
Attribute	Status node event register.
Usage	<p>Reads the status node event register:</p> <pre>nodeeventreg = status.node_event</pre>
Remarks	<p>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.</p> <p>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.</p> <p>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 (on page 13-264) register.</p>

status.node_event			
Example	<p>Reads the status node event register:</p> <pre>nodeeventreg = status.node_event print(nodeeventreg)</pre> <p>Output: 1.29000e+02</p> <p>The above output indicates that bits B0 (MSB) and B7 (OSB) are set.</p>		
<table border="0" style="width: 100%;"> <tr> <td style="width: 70%;">status.operation.*</td> <td style="width: 30%; text-align: right;"> .condition .enable .event .ntr .ptr </td> </tr> </table>		status.operation.*	.condition .enable .event .ntr .ptr
status.operation.*	.condition .enable .event .ntr .ptr		
Attribute	Operation event status register set.		
Usage	<p>To read condition, enable, event, NTR and PTR registers:</p> <pre>operreg = status.operation.condition operreg = status.operation.enable operreg = status.operation.event operreg = status.operation.ntr operreg = status.operation.ptr</pre> <p>To write to enable, NTR and PTR registers:</p> <pre>status.operation.enable = operreg status.operation.ntr = operreg status.operation.ptr = operreg</pre> <p>Set operreg to one of the following values:</p> <ul style="list-style-type: none"> • 0 clears all bits. • To set CAL bit (B0): <pre>status.operation.CALIBRATING</pre> or <pre>status.operation.CAL</pre> • To set MEAS bit (B4): <pre>status.operation.MEASURING</pre> or <pre>status.operation.MEAS</pre> • To set PRMPTS bit (B11): <pre>status.operation.PROMPTS</pre> - or - <pre>status.operation.PRMPTS</pre> • To set USER bit (B12): <pre>status.operation.USER</pre> • To set PROG bit (B14): <pre>status.operation.PROGRAM_RUNNING</pre> - or - <pre>status.operation.PROG</pre> 		

status.operation.*		.condition .enable .event .ntr .ptr
Remarks	<p>operreg can also be set to the decimal weight of the bit to be set.</p> <p>To set bit B0 (CAL), set operreg to 1 (2^0).</p> <p>To set bit B4 (MEAS), set operreg to 16 (2^4).</p> <p>To set bit B11 (PRMPTS), set operreg to 2048 (2^{11}).</p> <p>To set bit B12 (USER), set operreg to 4096 (2^{12}).</p> <p>To set bit B14 (PROG), set operreg to 16384 (2^{14}).</p> <p>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).</p>	
Details	<p>These attributes are used to read or write to the operation event registers.</p> <p>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.</p> <p>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.</p> <p>The used bits of the operation event registers are described as follows:</p> <ul style="list-style-type: none"> • 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. 	
Example	<p>Sets the MEAS bit of the operation enable register:</p> <pre>status.operation.enable = status.operation.MEAS</pre>	

status.operation.user.*		.condition .enable .event .ntr .ptr
Attribute	Operation user event register set.	
Usage	<p>Reads condition, enable, event, NTR and PTR registers:</p> <pre>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</pre> <p>Writes to condition, enable, NTR and PTR registers:</p> <pre>status.operation.user.enable = operreg status.operation.user.ntr = operreg status.operation.user.ptr = operreg</pre> <p>Set operreg to one of the following values:</p> <ul style="list-style-type: none"> • To clear all bits: 0 • To set user BIT0: status.operation.user.BIT0 • To set user BIT1: status.operation.user.BIT1 • To set user BIT2: status.operation.user.BIT2 • To set user BIT3: status.operation.user.BIT3 • To set user BIT4: status.operation.user.BIT4 • To set user BIT5: status.operation.user.BIT5 • To set user BIT6: status.operation.user.BIT6 • To set user BIT7: status.operation.user.BIT7 • To set user BIT8: status.operation.user.BIT8 • To set user BIT9: status.operation.user.BIT9 • To set user BIT10: status.operation.user.BIT10 • To set user BIT11: status.operation.user.BIT11 • To set user BIT12: status.operation.user.BIT12 • To set user BIT13: status.operation.user.BIT13 • To set user BIT14: status.operation.user.BIT14 	

<p>status.operation.user.*</p> <p>.condition .enable .event .ntr .ptr</p>	
	<p>operreg can also be set to the decimal weight of the bit to be set. To set bit X where X = 1 to 14, set operreg to 2^X. For example:</p> <ul style="list-style-type: none"> • To set user BIT0, set operreg to 1 (2⁰). • To set user BIT4, set operreg to 16 (2⁴). • To set user BIT11, set operreg to 2048 (2¹¹). <p>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).</p>
Remarks	<ul style="list-style-type: none"> • These attributes are 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 will set B1 (Bit 1) of the user event register: <ul style="list-style-type: none"> • status.operation.user.enable = 2 • status operation user condition register equals 2 or has bit 2 set. • 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. <p>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.</p>
Example	<p>Sets user BIT0 of the operation user enable register:</p> <pre>status.operation.user.enable = status.operation.user.BIT0</pre>

status.questionable.*		.condition .enable .event .ntr .ptr
Attribute	Questionable event status register set.	
Usage	<p>To read condition, enable, event, NTR, and PTR registers:</p> <pre>quesreg = status.questionable.condition quesreg = status.questionable.enable quesreg = status.questionable.event quesreg = status.questionable.ntr quesreg = status.questionable.ptr</pre> <p>To write to enable, NTR and PTR registers:</p> <pre>status.questionable.enable = quesreg status.questionable.ntr = quesreg status.questionable.ptr = quesreg</pre> <p>Set quesreg to one of the following values:</p> <ul style="list-style-type: none"> • 0 clears all bits • To set slot interlocks (B1-B6): status.questionable.n where n = 2, 4, 8, 16, 32, or 64 (for Slots 1 through 6, respectively). • To set DMM (B7): status.questionable.DMM or status.questionable.128 • To set CAL bit (B8): status.questionable.CALIBRATION or status.questionable.CAL or status.questionable.256 • To set thermal bit (B9-B13): status.questionable.n where n = 2, 4, 8, 16, 32, or 64 (for Slots 1 through 6, respectively). • For slot interlocks bits 1 to 6, for Slots 1 to 6 respectively, set either SLOTx_INTERLOCK or SxINL where x = 1 to 6, to indicate the interlock connection of a card in Slot x is in question • For DMM bit 7, set either DMM_CONNECTION or DMMCON to indicate that the DMM connection is in question for a measurement taken • For calibration bit 8, set either CALIBRATION or CAL to indicate that the calibration of the instrument is in question • For thermal bits 9 to 14 for Slots 1 to 6, respectively, set either SLOTx_THERMAL or SxTHR, where x = 1 to 6, to indicate that the thermal aspect of the card in slot x is in question. 	

	<p>status.questionable.*</p> <p>.condition .enable .event .ntr .ptr</p>
<p>Remarks</p>	<p>These attributes are used to read or write to the questionable status registers.</p> <p>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.</p> <p>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 (S4THR, Slot 4 thermal bit) are set.</p> <p>The used bits of the questionable event registers are described as follows:</p> <ul style="list-style-type: none"> • Bit B1, S1INL - Set bit indicates that the interlock connection of a card in Slot 1 is in question. • Bit B2, S2INL - Set bit indicates that the interlock connection of a card in Slot 2 is in question. • Bit B3, S3INL - Set bit indicates that the interlock connection of a card in Slot 3 is in question. • Bit B4, S4INL - Set bit indicates that the interlock connection of a card in Slot 4 is in question. • Bit B5, S5INL - Set bit indicates that the interlock connection of a card in Slot 5 is in question. • Bit B6, S6INL - Set bit indicates that the interlock connection of a card in Slot 6 is in question. • Bit B7, DMMCON - Set bit indicates that the DMM connection is in question for a measurement taken. • Bit B8, CAL - Set bit indicates that the calibration of the instrument is in question • Bit B9, S1THR - Set bit indicates that the thermal aspect of a card in Slot 1 is in question. • Bit B10, S2THR - Set bit indicates that the thermal aspect of a card in Slot 2 is in question. • Bit B11, S3THR - Set bit indicates that the thermal aspect of a card in Slot 3 is in question. • Bit B12, S4THR - Set bit indicates that the thermal aspect of a card in Slot 4 is in question. • Bit B13, S5THR - Set bit indicates that the thermal aspect of a card in Slot 5 is in question. • Bit B14, S6THR - Set bit indicates that the thermal aspect of a card in Slot 6 is in question.

status.questionable.*		.condition .enable .event .ntr .ptr
Details	<p>quesreg can be set to the decimal weight of the bit to be set:</p> <p>To set bit B1 (S1INL), set quesreg to 1 (2^1).</p> <p>To set bit B2 (S2INL), set quesreg to 4 (2^2).</p> <p>To set bit B3 (S3INL), set quesreg to 8 (2^3).</p> <p>To set bit B4 (S4INL), set quesreg to 16 (2^4).</p> <p>To set bit B5 (S5INL), set quesreg to 32 (2^5).</p> <p>To set bit B6 (S6INL), set quesreg to 64 (2^6).</p> <p>To set bit B7 (DMMCON), set quesreg to 128 (2^7).</p> <p>To set bit B8 (CAL), set quesreg to 256 (2^8).</p> <p>To set bit B9 (S1THR), set quesreg to 512 (2^9).</p> <p>To set bit B10 (S2THR), set quesreg to 1024 (2^{10}).</p> <p>To set bit B11 (S3THR), set quesreg to 2048 (2^{11}).</p> <p>To set bit B12 (S4THR), set quesreg to 4096 (2^{12}).</p> <p>To set bit B13 (S5THR), set quesreg to 8192 (2^{13}).</p> <p>To set bit B14 (S6THR), set quesreg to 16384 (2^{14}).</p> <p>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).</p>	
Example	<p>To set the CAL bit of the questionable enable register:</p> <pre>status.questionable.enable = status.questionable.CAL</pre>	

status.request_enable	
Attribute	Service request enable register.
Usage	<p>Reads service request enable register: <code>servenabreg = status.request_enable</code></p> <p>Writes to system enable register: <code>status.request_enable = servenabreg</code></p> <p>Set servenabreg to one of the following values:</p> <ul style="list-style-type: none"> • 0 Clears all bits. • To set (enables) MSB bit (B0): <code>status.MEASUREMENT_SUMMARY_BIT</code> - or - <code>status.MSB</code> • To set (enables) SSB bit (B1): <code>status.SYSTEM_SUMMARY_BIT</code> - or - <code>status.SSB</code> • To set (enables) EAV bit (B2): <code>status.ERROR_AVAILABLE</code> - or - <code>status.EAV</code> • To set (enables) QSB bit (B3): <code>status.QUESTIONABLE_SUMMARY_BIT</code> - or - <code>status.QSB</code> • To set (enables) MAV bit (B4): <code>status.MESSAGE_AVAILABLE</code> - or - <code>status.MAV</code> • To set (enables) ESB bit (B5): <code>status.EVENT_SUMMARY_BIT</code> - or - <code>status.ESB</code> • To set (enables) OSB bit (B7): <code>status.OPERATION_SUMMARY_BIT</code> - or - <code>status.OSB</code> <p>servenabreg can also be set to the decimal weight of the bit to be set: 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).</p>

status.request_enable	
Remarks	<ul style="list-style-type: none"> 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 (on page 13-264) register.
Example	Sets the MSB bit of the service request enable register: <pre>status.request_enable = status.MSB</pre>

status.request_event	
Attribute	Service request event register.
Usage	Reads the service request event register: <pre>serveventreg = status.request_event</pre>
Remarks	<p>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.</p> <p>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.</p> <p>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 (on page 13-264) register.</p>
Example	Reads the service request event register: <pre>serveventreg = status.request_event print(serveventreg)</pre> Output: 1.29000e+02 The above output indicates that bits B0 (MSB) and B7 (OSB) are set.

status.reset()	
Function	Resets all bits set in the status model.
Usage	<pre>status.reset()</pre>
Remarks	This function clears all status data structure registers (enable, event, NTR and PTR) to their power up states.

status.standard.*		.condition .enable .event
Attribute	Standard event register set.	
Usage	<p>Reads condition, enable and event registers:</p> <pre>standardreg = status.standard.condition standardreg = status.standard.enable standardreg = status.standard.event</pre> <p>Writes to enable register:</p> <pre>status.standard.enable = standardreg</pre> <p>Set standardreg to one of the following values:</p> <p>0 Clears all bits.</p> <p>To set the OPC bit (B0):</p> <pre>status.standard.OPERATION_COMPLETE</pre> <p>- or - <code>status.standard.OPC</code></p> <p>To set QYE bit (B2): <code>status.standard.QUERY_ERROR</code></p> <p>- or - <code>status.standard.QYE</code></p> <p>To set DDE bit (B3): <code>status.standard.DEVICE_DEPENDENT_ERROR</code></p> <p>- or - <code>status.standard.DDE</code></p> <p>To set EXE bit (B4): <code>status.standard.EXECUTION_ERROR</code></p> <p>- or - <code>status.standard.EXE</code></p> <p>To set CME bit (B5): <code>status.standard.COMMAND_ERROR</code></p> <p>- or - <code>status.standard.CME</code></p> <p>To set URQ bit (B6): <code>status.standard.USER_REQUEST</code></p> <p>- or - <code>status.standard.URQ</code></p> <p>To set PON bit (B7): <code>status.standard.POWER_ON</code></p> <p>- or - <code>status.standard.PON</code></p> <p>standardreg can also be set to the decimal weight of the bit to be set:</p> <p>To set bit B0 (OPC), set standardreg to 1 (2^0).</p> <p>To set bit B2 (QYE), set standardreg to 4 (2^2).</p> <p>To set bit B5 (CME), set standardreg to 32 (2^5).</p> <p>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).</p>	

status.standard.*		.condition .enable .event
Remarks	<p>These attributes are used to read or write to the standard status registers.</p> <p>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 7.</p> <p>For example, assume value 9 is returned for the enable register. The binary equivalent is 00001001. This value indicates that bit 0 (OPC) and bit 3 (DDE) are set.</p> <p>The used bits of the standard event status register are described as follows:</p> <ul style="list-style-type: none"> • Bit B0, Operation Complete (OPC) - Set bit indicates that all pending selected device operations are completed and the Series 3700 is ready to accept new commands. The bit is set in response to an *OPC command. The ICL function <code>opc()</code> 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 Series 3700 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: <ul style="list-style-type: none"> -IEEE-488.2 syntax error -- Series 3700 received a message that does not follow the defined syntax of the IEEE-488.2 standard. -Semantic error -- Series 3700 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 Series 3700 front panel was pressed. • Bit B7, Power ON (PON) - Set bit indicates that the Series 3700 has been turned off and turned back on since the last time this register was read. 	
Example	<p>Sets the PON bit of the standard event enable register:</p> <pre>status.standard.enable = status.standard.PON</pre>	

status.system.*		.condition .enable .event																
Attribute	TSP-Link™ system data structure register set.																	
Usage	<p>To read condition, enable and event registers:</p> <pre>enablereg = status.system.condition enablereg = status.system.enable enablereg = status.system.event</pre> <p>To write to enable register:</p> <pre>status.system.enable = enablereg</pre> <p>Set enablereg to one of the following values:</p> <p>0 Clears all bits.</p> <p>To set EXT bit (B0):</p> <pre>1 or status.system.EXTENSION_BIT</pre> <p>- or -</p> <pre>1 or status.system.EXT</pre> <p>To set a node bit (Bn); n = 1 to 14.</p> <pre>status.system.NODEn</pre>																	
Remarks	<ul style="list-style-type: none"> 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: <table border="1" data-bbox="540 1234 1302 1421"> <tr> <td>B0 - EXT bit</td> <td>B4 - Node 4</td> <td>B8 - Node 8</td> <td>B12 - Node 12</td> </tr> <tr> <td>B1 - Node 1</td> <td>B5 - Node 5</td> <td>B9 - Node 9</td> <td>B13 - Node 13</td> </tr> <tr> <td>B2 - Node 2</td> <td>B6 - Node 6</td> <td>B10 - Node 10</td> <td>B14 - Node 14</td> </tr> <tr> <td>B3 - Node 3</td> <td>B7 - Node 7</td> <td>B11 - Node 11</td> <td>B15 - Not used</td> </tr> </table> <ul style="list-style-type: none"> For example, assume value 9 is returned for the enable register. The binary equivalent is 000000000001001. 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. 		B0 - EXT bit	B4 - Node 4	B8 - Node 8	B12 - Node 12	B1 - Node 1	B5 - Node 5	B9 - Node 9	B13 - Node 13	B2 - Node 2	B6 - Node 6	B10 - Node 10	B14 - Node 14	B3 - Node 3	B7 - Node 7	B11 - Node 11	B15 - Not used
B0 - EXT bit	B4 - Node 4	B8 - Node 8	B12 - Node 12															
B1 - Node 1	B5 - Node 5	B9 - Node 9	B13 - Node 13															
B2 - Node 2	B6 - Node 6	B10 - Node 10	B14 - Node 14															
B3 - Node 3	B7 - Node 7	B11 - Node 11	B15 - Not used															
Also see	<p>status.system2.* (on page 13-281)</p> <p>status.system3.* (on page 13-282)</p> <p>status.system4.* (on page 13-284)</p> <p>status.system5.* (on page 13-285)</p>																	
Example	<p>Sets the extension bit of the system enable register:</p> <pre>status.system.enable = status.system.EXT</pre>																	

status.system2.*		.condition .enable .event																
Attribute	TSP-Link™ system2 data structure register set.																	
Usage	<p>Reads condition, enable and event registers:</p> <pre>enablereg = status.system2.condition enablereg = status.system2.enable enablereg = status.system2.event</pre> <p>Writes to enable register:</p> <pre>status.system2.enable = enablereg</pre> <p>Set enablereg to one of the following values:</p> <p>0 Clears all bits.</p> <p>To set EXT bit (B0):</p> <pre>1 or status.system2.EXTENSION_BIT</pre> <p>- or -</p> <pre>1 or status.system2.EXT</pre> <p>To set node bit (Bn); n = 15 to 28.</p> <pre>status.system2.NODEn</pre>																	
Remarks	<ul style="list-style-type: none"> 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: <table border="1" data-bbox="540 1234 1300 1421"> <tbody> <tr> <td>B0 - EXT bit</td> <td>B4 - Node 18</td> <td>B8 - Node 22</td> <td>B12 - Node 26</td> </tr> <tr> <td>B1 - Node 15</td> <td>B5 - Node 19</td> <td>B9 - Node 23</td> <td>B13 - Node 27</td> </tr> <tr> <td>B2 - Node 16</td> <td>B6 - Node 20</td> <td>B10 - Node 24</td> <td>B14 - Node 28</td> </tr> <tr> <td>B3 - Node 17</td> <td>B7 - Node 21</td> <td>B11 - Node 25</td> <td>B15 - Not used</td> </tr> </tbody> </table> <ul style="list-style-type: none"> 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. 		B0 - EXT bit	B4 - Node 18	B8 - Node 22	B12 - Node 26	B1 - Node 15	B5 - Node 19	B9 - Node 23	B13 - Node 27	B2 - Node 16	B6 - Node 20	B10 - Node 24	B14 - Node 28	B3 - Node 17	B7 - Node 21	B11 - Node 25	B15 - Not used
B0 - EXT bit	B4 - Node 18	B8 - Node 22	B12 - Node 26															
B1 - Node 15	B5 - Node 19	B9 - Node 23	B13 - Node 27															
B2 - Node 16	B6 - Node 20	B10 - Node 24	B14 - Node 28															
B3 - Node 17	B7 - Node 21	B11 - Node 25	B15 - Not used															
Also see	<p>status.system.* (on page 13-280)</p> <p>status.system3.* (on page 13-282)</p> <p>status.system4.* (on page 13-284)</p> <p>status.system5.* (on page 13-285)</p>																	
Example	<p>Sets the extension bit of the system2 enable register:</p> <pre>status.system2.enable = status.system2.EXT</pre>																	

status.system3.*		.condition .enable .event																
Attribute	TSP-Link™ system3 data structure register set.																	
Usage	<p>To read condition, enable and event registers: <code>enablereg = status.system3.condition</code> <code>enablereg = status.system3.enable</code> <code>enablereg = status.system3.event</code></p> <p>To write to enable register: <code>status.system3.enable = enablereg</code></p> <p>Set <code>enablereg</code> to one of the following values: 0 Clears all bits.</p> <p>To set EXT bit (B0) 1 or <code>status.system3.EXTENSION_BIT</code> 1 or <code>status.system3.EXT</code></p> <p>To set a node bit (Bn); n = 29 to 42. <code>status.system3.NODEn</code></p>																	
Remarks	<ul style="list-style-type: none"> 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: <table border="1" data-bbox="540 1188 1278 1493"> <tr> <td>B0 - EXT</td> <td>B4 - Node 32</td> <td>B8 - Node 36</td> <td>B12 - Node 40</td> </tr> <tr> <td>B1 - Node 29</td> <td>B5 - Node 33</td> <td>B9 - Node 37</td> <td>B13 - Node 41</td> </tr> <tr> <td>B2 - Node 30</td> <td>B6 - Node 34</td> <td>B10 - Node 38</td> <td>B14 - Node 42</td> </tr> <tr> <td>B3 - Node 31</td> <td>B7 - Node 35</td> <td>B11 - Node 39</td> <td>B15 - Not used</td> </tr> </table> <ul style="list-style-type: none"> For example, assume value 9 is returned for the enable register. The binary equivalent is 000000000001001. This value indicates that bit 0 (EXT) and bit 3 (Node 31) are set. Assigning a value to the <code>status.system3.enable</code> attribute sets the extension bit or a node bit of the system3 node enable register. 		B0 - EXT	B4 - Node 32	B8 - Node 36	B12 - Node 40	B1 - Node 29	B5 - Node 33	B9 - Node 37	B13 - Node 41	B2 - Node 30	B6 - Node 34	B10 - Node 38	B14 - Node 42	B3 - Node 31	B7 - Node 35	B11 - Node 39	B15 - Not used
B0 - EXT	B4 - Node 32	B8 - Node 36	B12 - Node 40															
B1 - Node 29	B5 - Node 33	B9 - Node 37	B13 - Node 41															
B2 - Node 30	B6 - Node 34	B10 - Node 38	B14 - Node 42															
B3 - Node 31	B7 - Node 35	B11 - Node 39	B15 - Not used															
Also see	status.system.* (on page 13-280) status.system2.* (on page 13-281) status.system4.* (on page 13-284) status.system5.* (on page 13-285)																	

status.system3.*		.condition .enable .event																
Example	Sets the extension bit of the system3 enable register: <code>status.system3.enable = status.system3.EXT</code>																	
status.system4.*		.condition .enable .event																
Attribute	TSP-Link™ system4 data structure register set.																	
Usage	<p>To read condition, enable and event registers: <code>enablereg = status.system4.condition</code> <code>enablereg = status.system4.enable</code> <code>enablereg = status.system4.event</code></p> <p>To write to enable register: <code>status.system4.enable = enablereg</code></p> <p>Set enablereg to one of the following values: 0 Clears all bits.</p> <p>To set EXT bit (B0): 1 or <code>status.system4.EXTENSION_BIT</code> - or - 1 or <code>status.system4.EXT</code></p> <p>To set a node bit (Bn); n = 43 to 56. <code>status.system4.NODEn</code></p>																	
Remarks	<ul style="list-style-type: none"> 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: <table border="1" data-bbox="540 1451 1278 1635"> <tr> <td>B0-EXT bit</td> <td>B4-Node 46</td> <td>B8-Node 50</td> <td>B12-Node 54</td> </tr> <tr> <td>B1-Node43</td> <td>B5-Node47</td> <td>B9-Node51</td> <td>B13-Node55</td> </tr> <tr> <td>B2-Node44</td> <td>B6-Node48</td> <td>B10-Node52</td> <td>B14-Node56</td> </tr> <tr> <td>B3-Node45</td> <td>B7-Node49</td> <td>B11-Node53</td> <td>B15-Not used</td> </tr> </table> <ul style="list-style-type: none"> For example, assume value 9 is returned for the enable register. The binary equivalent is 000000000001001. This value indicates that bit 0 (EXT) and bit 3 (Node 45) are set. Assigning a value to the <code>status.system4.enable</code> attribute sets the extension bit or a node bit of the system4 node enable register. 		B0-EXT bit	B4-Node 46	B8-Node 50	B12-Node 54	B1-Node43	B5-Node47	B9-Node51	B13-Node55	B2-Node44	B6-Node48	B10-Node52	B14-Node56	B3-Node45	B7-Node49	B11-Node53	B15-Not used
B0-EXT bit	B4-Node 46	B8-Node 50	B12-Node 54															
B1-Node43	B5-Node47	B9-Node51	B13-Node55															
B2-Node44	B6-Node48	B10-Node52	B14-Node56															
B3-Node45	B7-Node49	B11-Node53	B15-Not used															

status.system4.*		.condition .enable .event
Also see	status.system.* (on page 13-280) status.system2.* (on page 13-281) status.system3.* (on page 13-282) status.system5.* (on page 13-285)	
Example	Sets the extension bit of the system4 enable register: <code>status.system4.enable = status.system4.EXT</code>	
status.system5.*		.condition .enable .event
Attribute	TSP-Link™ system5 data structure register set.	
Usage	To read condition, enable and event registers: <code>enablereg = status.system5.condition</code> <code>enablereg = status.system5.enable</code> <code>enablereg = status.system5.event</code> To write to enable register: <code>status.system5.enable = enablereg</code> Set enablereg to one of the following values: 0 Clears all bits. To set EXT bit (B0): 1 or <code>status.system5.EXTENSION_BIT</code> - or - 1 or <code>status.system5.EXT</code> To set a node bit (Bn); n = 57 to 64. <code>status.system5.NODEn</code>	

status.system5.*		.condition .enable .event												
Remarks	<p>In an expanded system (TSP-Link), this attribute is used to read or write to the system5 node registers.</p> <p>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:</p> <table border="1"> <tr> <td>Bit B0 - EXT bit</td> <td>Bit B4 - Node 60</td> <td>Bit B8 - Node 64</td> </tr> <tr> <td>Bit B1 - Node 57</td> <td>Bit B5 - Node 61</td> <td>Bits B9 through B15 - Not used</td> </tr> <tr> <td>Bit B2 - Node 58</td> <td>Bit B6 - Node 62</td> <td></td> </tr> <tr> <td>Bit B3 - Node 59</td> <td>Bit B7 - Node 63</td> <td></td> </tr> </table> <p>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.</p> <p>Assigning a value to the status.system5.enable attribute sets the extension bit or a node bit of the system5 node enable register.</p>		Bit B0 - EXT bit	Bit B4 - Node 60	Bit B8 - Node 64	Bit B1 - Node 57	Bit B5 - Node 61	Bits B9 through B15 - Not used	Bit B2 - Node 58	Bit B6 - Node 62		Bit B3 - Node 59	Bit B7 - Node 63	
Bit B0 - EXT bit	Bit B4 - Node 60	Bit B8 - Node 64												
Bit B1 - Node 57	Bit B5 - Node 61	Bits B9 through B15 - Not used												
Bit B2 - Node 58	Bit B6 - Node 62													
Bit B3 - Node 59	Bit B7 - Node 63													
Also see	<p>status.system.* (on page 13-280)</p> <p>status.system2.* (on page 13-281)</p> <p>status.system3.* (on page 13-282)</p> <p>status.system4.* (on page 13-284)</p>													
Example	<p>Sets the extension bit of the system5 enable register:</p> <pre>status.system5.enable = status.system5.EXT</pre>													

timer functions

Use the functions in this group to control the timer. The timer can be used to measure the time it takes to perform various operations. Use the [timer.reset\(\)](#) (on page 13-287) function at the beginning of an operation to reset the timer to zero, and then use the [timer.measure.t\(\)](#) (on page 13-286) 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	<pre>time = timer.measure.t()</pre> <p>time: Returns the elapsed time in seconds. (1µs resolution).</p>

timer.measure.t()	
Remarks	<ul style="list-style-type: none"> 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() (on page 13-287)
Example	Resets the timer and then measures the time since the reset: <pre>timer.reset() ... time = timer.measure.t() print(time)</pre> Output: 1.469077e+01 The above output indicates that timer.measure.t was executed 14.69077 seconds after timer.reset.

timer.reset()	
Function	Resets the timer to 0 seconds.
Usage	<code>timer.reset()</code>
Remarks	This function will restart the timer at zero.
Also see	timer.measure.t() (on page 13-286)
Example	Resets the timer and then measures the time since the reset: <pre>timer.reset() ... time = timer.measure.t() print(time)</pre> Output: 1.469077e+01 The above output indicates that timer.measure.t was executed 14.69077 seconds after timer.reset.

trigger functions and attributes

Use the functions in this group to control triggering.

trigger.blender[N].clear()	
Function	This function clears and resets blender N.
Usage	<code>trigger.blender[N].clear()</code> N: The trigger line (1–2).

trigger.blender[N].orenable	
Attribute	Set orenable attribute for the blender.
Usage	To read the trigger blender orenable mode: <code>orenable = trigger.blender[N].orenable</code> To write the trigger blender orenable mode: <code>trigger.blender[N].orenable = orenable</code> orenable: The desired orenable mode (true/false) N: The trigger blender (1–2).
Remarks	This attribute selects whether the blender will wait for any one event (or-mode) or will wait for all selected events (and-mode) before signaling an output event. Set this attribute to true for or -mode. Set this attribute to false for and -mode.

trigger.blender[N].overrun	
Attribute	This attribute provides the event detector overrun status for a blender.
Usage	<code>overrun = trigger.blender[N].overrun</code> N: The trigger blender (1–2). overrun: Trigger overrun state for blender.
Remarks	This attribute is a read-only attribute that indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector built into the event blender itself. This attribute does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the event. It also is not an indication of an event overrun.

trigger.blender[N].stimulus[M]	
Attribute	This attribute provides event detector trigger selection.
Usage	To read blender and trigger stimulus event: <code>eventid = trigger.blender[N].stimulus[M]</code> To write blender and trigger stimulus event: <code>trigger.blender[N].stimulus[M] = eventid</code> N: Event blender number (1–2). M: Trigger stimulus event (1-4). eventid: Event to trigger the stimulus action.

trigger.blender[N].stimulus[M]	
Remarks	<p>This attribute selects which events will trigger the blender. There are 4 acceptors that can each select a different event.</p> <p>eventid may be one of the following (existing trigger event IDs):</p> <ul style="list-style-type: none"> • digio.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line. • display.trigger.EVENT_ID: The trigger key on the front panel is pressed. • trigger.EVENT_ID: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event. • trigger.blender[N].EVENT_ID: A combination of events has occurred. • trigger.timer[N].EVENT_ID: A delay expired. • tsplink.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line. • lan.trigger[N].EVENT_ID • scan.trigger.EVENT_SCAN_READY: Scan Ready Event. • scan.trigger.EVENT_SCAN_START: Scan Start Event • scan.trigger.EVENT_CHANNEL_READY: Channel Ready Event • scan.trigger.EVENT_MEASURE_COMP: Measure Complete Event • scan.trigger.EVENT_SEQUENCE_COMP: Sequence Complete Event • scan.trigger.EVENT_SCAN_COMP: Scan Complete Event • scan.trigger.EVENT_IDLE: Idle Event <hr/> <p>NOTE Use the ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because they may need to change when enhancements are added to the instrument.</p>

trigger.blender[N].wait()	
Function	This function waits for a blender trigger event to occur.
Usage	<pre>triggered = trigger.blender[N].wait(timeout)</pre> <p>N: The trigger blender (1–2).</p> <p>timeout: Maximum amount of time in seconds to wait for the trigger blender event.</p> <p>triggered: Trigger detection indication for blender.</p>
Remarks	<p>This function will wait for an event blender trigger event. If one or more trigger events were detected since the last time <code>trigger.blender[N].wait</code> or trigger.blender[N].clear() (on page 13-287) was called, this function will return immediately.</p> <p>After detecting a trigger with this function, the event detector will automatically reset and rearm. This is true regardless of the number of events detected.</p>

trigger.clear()	
Function	Clears the command interface trigger event detector.
Usage	<code>trigger.clear()</code>

trigger.clear()	
Remarks	The trigger event detector remembers if an event has been detected since the last trigger.wait() (on page 13-290) call. This function clears the trigger's event detector and discards the previous history of command interface trigger events.
Also see	trigger.wait() (on page 13-290)
trigger.wait()	
Function	Wait for a trigger event.
Usage	<code>triggered = trigger.wait(timeout)</code> 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	<ul style="list-style-type: none"> 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.
Also see	trigger.clear() (on page 13-289)
Example	<p>Waits up to 10 seconds for a trigger:</p> <pre>triggered = trigger.wait(10) print(triggered)</pre> <p>Output: false</p> <p>The above output indicates that no trigger was detected during the 10 second timeout.</p>

trigger.timer functions and attributes

Use the functions in this group to control the trigger timer.

trigger.timer[N].clear	
Attribute	Clears trigger time[N].
Usage	<code>trigger.timer[N].clear</code> N: is a trigger.timer value (from 1–4).
Remarks	This function clears the specified trigger timer number.
Also see	trigger.timer[N].count (on page 13-290)

trigger.timer[N].count	
Attribute	Retrigger count.
Usage	To read: <code>count = trigger.timer[N].count</code> To write: <code>trigger.timer[N].count = count</code> N: is a trigger timer value (from 1–4). count: Repeat trigger count.
Remarks	This attribute sets the number of times the timer will trigger an event. If this attribute is set greater than 1, the timer will automatically start the next delay at expiration of a previous delay.
Also see	trigger.timer[N].clear (on page 13-290)
Example	To read retrigger count for timer number 1: <code>print(trigger.timer[1].count)</code>

trigger.timer[N].delay	
Attribute	The timer interval used when triggered.
Usage	To read delay interval: <code>interval = trigger.timer[N].delay</code> To write delay interval: <code>trigger.timer[N].delay = interval</code> N: is a trigger timer value (from 1–4). interval: Delay interval in seconds.
Remarks	This attribute sets a fixed timer delay. Each time the timer is triggered it will use this delay period.

trigger.timer[N].delaylist	
Attribute	An array of timer intervals used when triggered.
Usage	To read array of timer intervals: <code>intervals = trigger.timer[n].delay</code> To write array of timer intervals: <code>trigger.timer[n].delay = intervals</code> n: (In) Trigger timer number. intervals: (In/Out) Table of delay intervals in seconds.

trigger.timer[N].delaylist	
Remarks	This attribute sets an array of timer delays. Each time the timer is triggered it will use the next delay period from the array. After all elements in the array have been used, the last element will be used for subsequent triggers.
trigger.timer[N].overrun	
Attribute	Event detector overrun status.
Usage	<pre>overrun = trigger.timer[N].overrun</pre> <p>N: is a trigger timer value (from 1–4).</p> <p>overrun: Trigger overrun state.</p>
Remarks	This attribute is a read-only attribute that indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.
trigger.timer[N].passthrough	
Attribute	Trigger pass-through enable.
Usage	<p>To read pass-through state:</p> <pre>passthrough = trigger.timer[N].passthrough</pre> <p>To write pass-through state:</p> <pre>trigger.timer[N].passthrough = passthrough</pre> <p>N: is a trigger timer value (from 1–4).</p> <p>passthrough: Pass-through true/false.</p>
Remarks	This attribute enables or disables the timer trigger's pass-through mode. When enabled, triggers are passed through immediately as well as initiating the delay. When disabled, a trigger will only initiate a delay. passthrough can be either true or false.
trigger.timer[N].stimulus	
Attribute	Event to cause the delay to start.
Usage	<p>To read the stimulus event:</p> <pre>eventid = trigger.timer[N].stimulus</pre> <p>To write the stimulus event:</p> <pre>trigger.timer[N].stimulus = eventid</pre> <p>N: is a trigger timer value (from 1–4).</p> <p>eventid: Event to trigger the timer delay.</p>

trigger.timer[N].stimulus	
Remarks	<p>This attribute selects which event will start the timer.</p> <p>eventid may be one of the following (existing trigger event IDs):</p> <ul style="list-style-type: none"> • digio.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line. • display.trigger.EVENT_ID: The trigger key on the front panel is pressed. • trigger.EVENT_ID: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event. • trigger.blender[N].EVENT_ID: A combination of events has occurred. • trigger.timer[N].EVENT_ID: A delay expired. • tsplink.trigger[N].EVENT_ID: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line. • lan.trigger[N].EVENT_ID • scan.trigger.EVENT_SCAN_READY: Scan Ready Event. • scan.trigger.EVENT_SCAN_START: Scan Start Event • scan.trigger.EVENT_CHANNEL_READY: Channel Ready Event • scan.trigger.EVENT_MEASURE_COMP: Measure Complete Event • scan.trigger.EVENT_SEQUENCE_COMP: Sequence Complete Event • scan.trigger.EVENT_SCAN_COMP: Scan Complete Event • scan.trigger.EVENT_IDLE: Idle Event <hr/> <p>NOTE Use the ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because it may need to change when enhancements are added to the instrument.</p>
Example	<p>To print the delay:</p> <pre>print(trigger.timer[1].stimulus)</pre>
trigger.timer[N].wait()	
Function	Wait for a trigger.
Usage	<pre>triggered = trigger.timer[N].wait(timeout)</pre> <p>N: is a trigger timer value (from 1–4).</p> <p>timeout: Maximum amount of time in seconds to wait for the trigger.</p> <p>triggered: Trigger detection indication.</p>
Remarks	<p>This function will wait for a timer trigger. If one or more trigger events were detected since the last time <code>trigger.timer[N].wait</code> or trigger.timer[N].clear (on page 13-290) was called, this function will return immediately.</p> <p>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.</p>

tsplink functions and attributes

Use the function and attributes in this group to assign node numbers to Series 3700 instruments and initialize the TSP-Link™ system.

tsplink.group	
Attribute	The group number of a TSP-Link™ node.
Usage	<pre>groupnumber = tsplink.group tsplink.group = groupnumber</pre> <p>groupnumber: The TSP-Link group number for the node.</p>
Remarks	This attribute controls the TSP-Link group number used for DTNS. Set this attribute to zero to remove the node from all subgroups.
tsplink.master	
Attribute	The node number of the master node.
Usage	<pre>master = tsplink.master</pre> <p>master: The node number of the master node.</p>
Remarks	This read-only attribute indicates which node is the master node.
tsplink.node	
Attribute	TSP-Link™ node number.
Usage	<p>To read the node number:</p> <pre>mynode = tsplink.node</pre> <p>To write the node number:</p> <pre>tsplink.node = mynode</pre> <p>mynode: Set node number from 1 to 64. Default value is 2.</p>
Remarks	<ul style="list-style-type: none"> • This attribute sets the TSP-Link node number and saves the value in nonvolatile memory. • After changing the node number, it will not take effect until the next time tsplink.reset() (on page 13-294) is executed on any node in the system. • Each node connected to the TSP-Link must be assigned a different node number.
Example	<p>Sets the TSP-Link node to number 2:</p> <pre>tsplink.node = 2</pre>
tsplink.reset()	
Function	Initializes (resets) all nodes (instruments) in the TSP-Link™ system.
Usage	<pre>tsplink.reset()</pre>

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.

tsplink.state	
Attribute	TSP-Link™ online state.
Usage	<code>state = tsplink.state</code>
Remarks	<ul style="list-style-type: none"> 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() (on page 13-294) is successful, the state will be "online". This attribute is read-only.
Example	Reads the state of the TSP-Link: <pre>state = tsplink.state print(state) → online</pre>

tsplink.trigger functions and attributes

The functions in this group control the TSP-Link™'s trigger event detector.

tsplink.trigger[N].assert()	
Function	Simulates the occurrence of the trigger and generates the corresponding event id.
Usage	<code>tsplink.trigger[N].assert()</code> N: The trigger line to assert (1–3).
Remarks	This function will generate a trigger pulse on the given TSP-Link™ trigger line.
Also see	tsplink.trigger[N].clear() (on page 3-13) tsplink.trigger[N].mode (on page 3-13) tsplink.trigger[N].overrun (on page 3-15) tsplink.trigger[N].release() (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)

tsplink.trigger[N].clear()	
Function	Clear the event detector for a trigger.
Usage	<code>tsplink.trigger[N].clear()</code> N: The trigger line (1–3).

tsplink.trigger[N].clear()	
Remarks	A trigger's event detector remembers if an event has been detected since the last <code>tsplink.trigger[N].wait</code> call. This function clears a trigger's event detector and discards the previous history of the trigger line.
Also see	tsplink.trigger[N].mode (on page 3-13) tsplink.trigger[N].overrun (on page 3-15) tsplink.trigger[N].release() (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)
tsplink.trigger[N].mode	
Attribute	The trigger operation/detection mode.
Usage	<p>To read trigger operation/detection mode: <code>mode = tsplink.trigger[N].mode</code></p> <p>To write trigger operation/detection mode: <code>tsplink.trigger[N].mode = mode</code></p> <p>N: The trigger line (1–3). mode: Trigger mode.</p>

tsplink.trigger[N].mode	
Remarks	<p>This attribute controls the mode in which the trigger event detector as well as the output trigger generator will operate on the given trigger line. mode can be one of the following values:</p> <p>tsplink.TRIG_BYPASS Allow direct control of the line.</p> <p>tsplink.TRIG_EITHER Detect rising or falling edge triggers as input. Assert a TTL-low pulse for output.</p> <p>tsplink.TRIG_FALLING Detect falling edge triggers as input. Assert a TTL-low pulse for output.</p> <p>tsplink.TRIG_RISING Use digio.TRIG_RISINGA if the line is in the high output state. Use digio.TRIG_RISINGM if the line is in the low output state.</p> <p>tsplink.TRIG_RISINGA Detect rising edge triggers as input. Assert a TTL-low pulse for output.</p> <p>tsplink.TRIG_RISINGM Assert a TTL-high pulse for output. Input edge detection is not possible in this mode.</p> <p>tsplink.TRIG_SYNCHRONOUS Detect falling edge triggers as input and latch them low. Assert a TTL-low pulse for output.</p> <p>tsplink.TRIG_SYNCHRONOUSA Detect falling edge triggers as input and automatically latch and drive them low when detected. Release a latched line for output.</p> <p>tsplink.TRIG_SYNCHRONOISM Detect rising edge triggers as input. Assert a TTL-low pulse for output.</p>
Remarks, continued	<p>The default trigger mode for a line will be TRIG_BYPASS. In this mode, the line can be directly controlled as a digital I/O line. When programmed to any other mode, the output state of the I/O line is controlled by the trigger logic and the user-specified output state of the line will be ignored.</p> <p>For compatibility with older firmware, when the trigger mode is set to TRIG_RISING, the user specified output state of the line will be examined. If the output state selected when the mode is changed is high, the actual mode used will be TRIG_RISINGA. If the output state selected when the mode is changed is low, the actual mode used will be TRIG_RISINGM.</p>

tsplink.trigger[N].mode	
Also see	tsplink.trigger[N].assert (on page 3-12) tsplink.trigger[N].clear() (on page 3-13) tsplink.trigger[N].overrun (on page 3-15) tsplink.trigger[N].release() (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)
tsplink.trigger[N].overrun	
Attribute	Event detector overrun status.
Usage	<pre>overrun = tsplink.trigger[N].overrun</pre> <p>overrun: Trigger overrun state. N: The trigger line (1–3).</p>
Remarks	<p>This attribute is a read-only attribute. It indicates if an event was ignored due to the event detector being in the detected state when the event occurred. This is an indication of the state of the event detector built into the synchronization line itself.</p> <p>This attribute does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the event.</p>
Also see	tsplink.trigger[N].assert (on page 3-12) tsplink.trigger[N].clear() (on page 3-13) tsplink.trigger[N].mode (on page 3-13) tsplink.trigger[N].release() (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)
tsplink.trigger[N].release()	
Function	Release a latched trigger.
Usage	<pre>tsplink.trigger[N].release()</pre> <p>N: The trigger line (1–3).</p>
Remarks	This function will release a latched trigger on the given TSP-Link™ trigger line.
Also see	tsplink.trigger[N].assert (on page 3-12) tsplink.trigger[N].clear() (on page 3-13) tsplink.trigger[N].mode (on page 3-13) tsplink.trigger[N].overrun (on page 3-15) tsplink.trigger[N].stimulus (on page 13-298) tsplink.trigger[N].wait() (on page 3-15)

tsplink.trigger[N].stimulus	
Attribute	Event to cause this trigger to assert.
Usage	<code>tsplink.trigger[N].stimulus()</code> N: The trigger line (1–3).
Remarks	<p>This attribute selects which event will cause the synchronization line to assert a trigger.</p> <p>eventid may be one of the following (existing trigger event IDs):</p> <ul style="list-style-type: none"> • <code>digio.trigger[N].EVENT_ID</code>: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line. • <code>display.trigger.EVENT_ID</code>: The trigger key on the front panel is pressed. • <code>trigger.EVENT_ID</code>: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event. • <code>trigger.blender[N].EVENT_ID</code>: A combination of events has occurred. • <code>trigger.timer[N].EVENT_ID</code>: A delay expired. • <code>tsplink.trigger[N].EVENT_ID</code>: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line. • <code>lan.trigger[N].EVENT_ID</code> • <code>scan.trigger.EVENT_SCAN_READY</code>: Scan Ready Event. • <code>scan.trigger.EVENT_SCAN_START</code>: Scan Start Event • <code>scan.trigger.EVENT_CHANNEL_READY</code>: Channel Ready Event • <code>scan.trigger.EVENT_MEASURE_COMP</code>: Measure Complete Event • <code>scan.trigger.EVENT_SEQUENCE_COMP</code>: Sequence Complete Event • <code>scan.trigger.EVENT_SCAN_COMP</code>: Scan Complete Event • <code>scan.trigger.EVENT_IDLE</code>: Idle Event <hr/> <p>NOTE Use the ICL define to set the stimulus value rather than the define value. Doing this will make the code compatible for future upgrades because it may need to change when enhancements are added to the instrument.</p>

tsplink.trigger[N].wait()	
Function	Wait for a trigger.
Usage	<code>triggered = tsplink.trigger[N].wait(timeout)</code> N: The trigger line (1–3). timeout: Maximum amount of time in seconds to wait for the trigger. triggered: Trigger detection indication.
Remarks	<p>This function will wait for an input trigger. If one or more trigger events were detected since the last time <code>tsplink.trigger[N].wait()</code> or <code>tsplink.trigger[N].clear()</code> (on page 3-13) was called, this function will return immediately.</p> <p>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.</p>

tspnet functions and attributes

Use the `tspnet` commands to control, identify, and communicate with TSP™ devices. You can use these commands with Keithley Instruments and non-Keithley Instruments devices.

tspnet.clear()	
Function	Device read clear buffer.
Usage	<code>tspnet.clear(<connection id>)</code> connection id: Integer value used as a handle for other <code>tspnet</code> commands
Remarks	This command clears any pending output data available from the device. No data is returned to the caller. No data is processed. Errors: <ul style="list-style-type: none"> Invalid Specified Connection
Example	<pre>tspnet.write(mydevice, 'print([[hello]])')</pre> <pre>print(tspnet.readavailable(mydevice))</pre> <p>Output</p> <pre>6.000000000e+000</pre> <pre>tspnet.clear(mydevice)</pre> <pre>print(tspnet.readavailable(mydevice))</pre> <p>Output</p> <pre>0.000000000e+000</pre>

tspnet.connect()	
Function	Device connection.
Usage	<p>To connect to any remote device on the LAN:</p> <pre><connection id> = tspnet.connect([<ip address>, [<port number>, <initialize string>]])</pre> <p>To connect to a TSP-enabled remote device on the LAN:</p> <pre><connection id> = tspnet.connect([<ip address>, [<password>]])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>ip address: String variable for passing the IP address</p> <p>port number: Optional integer value of the port number</p> <p>initialize string: String type for the initialization string to send</p>

tspnet.connect()	
Remarks	<p>This command connects a device to another device by way of the LAN interface (using the optionally-specified port number). The default port number is 5025. If the port number is 23, the interface will use the Telnet protocol (and set appropriate termination characters) to communicate with the device.</p> <p>If a port number and initialization string are provided, the remote device is assumed to be non-TSP-enabled. The Series 3700 does not perform any extra processing, prompt handling, error handling, or sending of commands. Additionally, the <code>tspnet.tsp</code> commands do not apply for use on this this remote device.</p> <p>If no port number and initialization string is provided, the remote device is assumed to be a Keithley Instruments TSP-enabled device. Depending on the state of <code>tspnet.tsp.abortonconnect</code> (on page 10-14), the Series 3700 sends an <code>abort()</code> to the remote device upon connection. The Series 3700 also enables TSP prompts on the remote device and error management. The Series 3700 places remote errors from the TSP-enabled device in its own error queue and prefaces these errors with "Remote Error", followed by an error description. Do not manually change either the prompt functionality (<code>localnode.prompts</code>) or show errors functionality (<code>localnode.showerrors</code>) on the remote TSP-enabled device, or subsequent <code>tspnet.tsp.*</code> commands using the connection may fail.</p> <p>You can simultaneous connect to a maximum of 32 remote devices.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Connection Failed • Connection Failed, Timeout • Invalid IP Address or Port Number
Example	<p>To connect to a TSP-enabled device:</p> <pre>mytspdevice = tspnet.connect('10.80.64.216')</pre> <p>To connect to a non-TSP-enabled device:</p> <pre>mydevice = tspnet.connect("192.168.1.51",1394, "*rst\r\n")</pre>

tspnet.disconnect()	
Function	Device disconnection.
Usage	<pre>tspnet.disconnect(<connection id>)</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p>
Remarks	<p>This command disconnects the two devices by closing the connection.</p> <p>For Keithley Instruments TSP™ devices, this results in any remotely running commands or scripts being aborted (terminated).</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection
Example	<pre>tspnet.disconnect(mydevice)</pre>

tspnet.execute()	
Function	Executes a command string on the remote device.
Usage	<pre>[variable =] tspnet.execute(<connection id>, <command string>, [<format string>])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>command string: Command to send to instrument.</p> <p>format string: Definition of format string for the input field using zeros (0), the decimal point (.), the polarity sign (+), and 'E' for exponent.</p>
Remarks	<p>This command sends the command string to the connection device. A termination is added to the command string when it is sent to the device (see <code>tspnet.termination()</code> (on page 10-11)). Optionally, when a format string is specified, the command waits for a string from the device. The Series 3700 decodes the output string according to the format specified in the format string and returns this output string as arguments from the function (see <code>tspnet.read()</code> (on page 10-8) for format specifiers).</p> <p>When this command is sent to a TSP-enabled device, the Series 3700 suspends operation until a timeout error is generated or until the device responds, even if no format string is specified. The TSP prompt from the remote device is read and thrown away. The Series 3700 places any remotely-generated errors into its error queue. When the optional format string is not specified, this command is equivalent to <code>tspnet.write()</code> (on page 10-7), except that a termination is automatically added to the end of the line.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Write Failed, Timeout • Write Failed • Read Failed, Timeout • Read Failed, Aborted • Read Failed • Remote Error, <remote error generated by command>
Example	<pre>Command remote device to run script named 'runmyscript()': tspnet.execute(mydevice, 'runmyscript()')</pre> <pre>Command remote device to execute a *idn?: tspnet.termination(mydevice, tspnet.TERM_CRLF) tspnet.execute(mydevice, '*idn?') print("instrument write/read returns:: " , tspnet.read(id_instr))</pre>

tspnet.idn()	
Function	Retrieves response of remote device to '*IDN?'
Usage	<pre><idn string> = tspnet.idn(<connection id>)</pre> <p>idn_string: Response as a string type</p> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p>
Remarks	<p>Sends the '*idn?' string to the remote device and retrieves its response.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Connection Not Available • Connection Failed, Aborted • Write Failed, Timeout • Write Failed • Read Failed, Timeout • Read Failed • Read Failed, Aborted
Example	<p>Retrieve and print response of 'IDN?*' from the remote device:</p> <pre>print(tspnet.idn(mydevice))</pre> <p>KEITHLEY INSTRUMENTS INC.,MODEL 3706,34345656,01.02a</p>

tspnet.read()	
Function	Reads data from remote device.
Usage	<pre>[variable =] tspnet.read(<connection id>, [<format string>])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>format string: Definition of format string for the input field using zeros (0), the decimal point (.), the polarity sign (+), and 'E' for exponent.</p>

tspnet.read()									
Remarks	<p>This command reads available data from the device (as indicated by the format string) and returns the number of arguments (as indicated by the format string).</p> <p>The format string can contain the following identifiers:</p> <table border="0"> <tr> <td><code>%[width]s</code></td> <td>Read data until the specific length</td> </tr> <tr> <td><code>%[max width]t</code></td> <td>Read data until the specific length or delimited by punctuation</td> </tr> <tr> <td><code>%[max width]n</code></td> <td>Read data until a newline and/or carriage return</td> </tr> <tr> <td><code>%d</code></td> <td>Read a number (delimited by punctuation)</td> </tr> </table> <p>If no format is specified, the command returns a string containing the data until a new line is reached. If no data is available, the Series 3700 will hold off operation until the requested data is available or until a timeout error is generated. Use <code>tspnet.timeout</code> to specify the timeout period.</p> <p>A maximum of 10 specifiers are allowed in a format string.</p> <p>When reading from a TSP-enabled remote device, the Series 3700 removes TSP prompts and places any errors received from the remote device into its own error queue. The Series 3700 prefaces errors from the remote device with "Remote Error," and followed by with the error number and error description.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Read Failed, Timeout • Read Failed, Aborted • Read Failed • Remote Error, <remote error generated by command> 	<code>%[width]s</code>	Read data until the specific length	<code>%[max width]t</code>	Read data until the specific length or delimited by punctuation	<code>%[max width]n</code>	Read data until a newline and/or carriage return	<code>%d</code>	Read a number (delimited by punctuation)
<code>%[width]s</code>	Read data until the specific length								
<code>%[max width]t</code>	Read data until the specific length or delimited by punctuation								
<code>%[max width]n</code>	Read data until a newline and/or carriage return								
<code>%d</code>	Read a number (delimited by punctuation)								
Example	<p>Send <code>"*idn?"</code> to remote device:</p> <pre>tspnet.write(id_instr, "*idn?\r\n")</pre> <p>Read and print response from remote device:</p> <pre>print("instrument write/read returns:: " , tspnet.read(id_instr))</pre>								

tspnet.readavailable()	
Function	Device read output available.
Usage	<p><code>[<num bytes> =] tspnet.readavailable(<connection id>)</code></p> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p>
Remarks	<p>This command checks to see if any output data is available from the device. No data is read. It is intended to allow TSP™ scripts to continue to run without waiting on a remote command to finish.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Read Failed

tspnet.readavailable()	
Example	<code>x = tspnet.readavailable(mydevice)</code>
tspnet.reset()	
Function	Device all disconnection.
Usage	<code>tspnet.reset()</code>
Remarks	<p>This command disconnects the all devices currently connected.</p> <p>For Keithley Instruments TSP™ devices, this results in any remotely running commands or scripts being terminated.</p> <p>Errors:</p> <ul style="list-style-type: none"> • <none>
Example	<code>tspnet.reset()</code>
tspnet.termination()	
Function	Device line termination.
Usage	<p><code><termination type> = tspnet.termination(<connection id>, [<termination type>])</code></p> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>termination type: <code>tspnet.TERM_LF</code>, <code>tspnet.TERM_CR</code>, <code>tspnet.TERM_CRLF</code>, or <code>tspnet.TERM_LFCR</code></p>
Remarks	<p>This setting sets and gets the termination characters used to determine the end of a line for lines being received by a connection. It also is used to terminate lines being sent to a connection. Pass the optional set value to set the termination. The current value is always returned. There are four possible values: LF, CR, CRLF, or LFCR. For TSP™ devices, the default is LF. For non-TSP devices, the default is CRLF. The termination character resets to default when a connection is terminated.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Invalid Termination
Example	<p>Set termination character:</p> <pre>tspnet.termination(mydevice, tspnet.TERM_LF)</pre> <p>Gets termination character and evaluates if set to LF. Response of "1" means true, set to <termination type>. Response of "0" means false, not set to <termination type>:</p> <pre>print(tspnet.termination(mydevice) == tspnet.TERM_LF)</pre> <p>Output:</p> <pre>1.0000000e+000</pre>

tspnet.timeout	
Attribute	Sets timeout value for <code>tspnet.connect()</code> , <code>tspnet.execute()</code> , and <code>tspnet.read()</code> commands.
Usage	<code>tspnet.timeout [= <seconds value>]</code> seconds value: Value in seconds
Remarks	This setting sets the duration the <code>tspnet.connect</code> , <code>tspnet.read</code> , and <code>tspnet.execute</code> commands will wait for a response. The time is specified in seconds. The default value is 5.0 seconds. The timeout may contain fractional seconds but is only accurate to the nearest 10mS. The timeout may be between 0.0 and 30 seconds. Errors: <ul style="list-style-type: none"> Invalid Timeout
Example	<code>tspnet.timeout = 10.0</code>

tspnet.tsp.abort()	
Function	Aborts device execution.
Usage	<code>tspnet.tsp.abort(<connection id>)</code> connection id: Integer value used as a handle for other <code>tspnet</code> commands
Remarks	This convenience command simply sends an "abort" string to a device. Errors: <ul style="list-style-type: none"> Invalid Specified Connection Connection Not Available Write Failed
Example	<code>tspnet.tsp.abort()</code>

tspnet.tsp.abortonconnect	
Attribute	Abort on connect.
Usage	<code>tspnet.tsp.abortonconnect [= <value>]</code> value: <code>tspnet.TRUE</code> or <code>tspnet.FALSE</code>

tspnet.tsp.abortonconnect	
Remarks	<p>This setting determines if the Series 3700 sends <code>abort()</code> when it attempts to connect using <code>tspnet.connect()</code> (on page 10-5) to a TSP™-enabled device. The default value is <code>tspnet.TRUE</code> (or non-zero).</p> <p>Sending the <code>abort()</code> command on connection causes any other active interfaces being used on that device to close to ensure you have obtained access to the remote device.</p> <p>Connecting to a TSP device without issuing an <code>abort()</code> command, or when <code>tspnet.tsp.abortonconnect</code> (on page 10-14) is set to <code>tspnet.FALSE</code>, can result in the Series 3700 suspending operation until it receives a response back from the device or until a timeout error generates.</p> <p>Errors:</p> <ul style="list-style-type: none"> • <none>
Example	<code>tspnet.tsp.abortonconnect = tspnet.FALSE</code>
tspnet.tsp.rhtablecopy()	
Function	Copies a reading buffer synchronous table from a device.
Usage	<pre><array> = tspnet.tsp.rhtablecopy(<connection id>, <name>, [<start index>, <end index>])</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>name: Parameter name from a listed group</p> <p>start index: Integer start value</p> <p>end index: Integer end value</p>
Remarks	<p>This convenience command reads the data from a reading buffer on a remote device and returns an array of numbers or a string representing the data. The name argument identifies the reading buffer name and synchronous table to copy. The optional start index and end index specify the portion of the reading buffer to read. If no index is specified, the entire buffer will be copied.</p> <p>This command is limited to transferring 50,000 readings at a time.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Write Failed, Timeout • Write Failed • Read Failed, Timeout • Read Failed, Aborted • Read Failed • Invalid Reading Buffer Table • Invalid Index Range • Out of Memory • Remote Error, <remote error generated by command>

tspnet.tsp.rhtablecopy()	
Example	<pre>table = tspnet.tsp.rhtablecopy(mytspdevice, 'myremotebuffername.readings', 1, 3) print(table[1], table[2], table[3])</pre> <p>Output:</p> <pre>4.5653423423e-1 4.5267523423e-1 4.5753543423e-1</pre> <pre>times = tspnet.tsp.rhtablecopy(mytspdevice, 'myremotebuffername.timestamps', 1, 3) print(times)</pre> <p>Output</p> <pre>01/01/2008 10:10:10.0000013,01/01/2008 10:10:10.0000233,01/01/2008 10:10:10.0000576</pre>
tspnet.tsp.runscript()	
Function	Load and runs a script on a device.
Usage	<pre>tspnet.tsp.runscript(<connection id>, [<name>], <script>)</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>name: Optional parameter name from a listed group</p> <p>script: The actual script itself as a string, enclosed in quotes</p>
Remarks	<p>This convenience command downloads a script to a device and runs it. It automatically adds the appropriate loadscript and endscript around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.</p> <p>The script is automatically loaded, compiled, and run. If there are no runnable lines (contains only functions), running has no effect.</p> <p>To load only and run at a later time, simply make sure the script contains only functions. Use <code>tspnet.execute()</code> to execute those functions at a later time.</p> <p>This command is appropriate only for TSP™-enabled devices.</p> <p>If no name is specified, one will be generated internally.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Write Failed, Timeout • Write Failed • Read Failed, Timeout • Read Failed, Aborted • Read Failed • Remote Error, <remote error generated by command>
Example	<pre>tspnet.tsp.runscript(mytspdevice, 'mytest', 'print([[start]]) for d = 1,10 do print([[work]]) end print([[end]])')</pre>

tspnet.write()	
Function	Write strings to remote device.
Usage	<pre>tspnet.write(<connection id>, <input string>)</pre> <p>connection id: Integer value used as a handle for other <code>tspnet</code> commands</p> <p>input string: String type used for writing to the remote instrument</p>
Remarks	<p>The <code>tspnet.write()</code> command sends the command string to the connection device. It does not wait for command completion on the remote device.</p> <p>The Series 3700 sends the input string to the remote device exactly as indicated. The input string must contain any necessary new lines, termination, or other indicators.</p> <p>Errors:</p> <ul style="list-style-type: none"> • Invalid Specified Connection • Write Failed, Timeout • Write Failed
Example	<p>Command remote device to run script named 'runmyscript':</p> <pre>tspnet.write(mydevice, 'runmyscript()\n')</pre> <p>Send a *idn? to a remote device:</p> <pre>tspnet.write(id_instr, "*idn?" .. "\r\n")</pre> <p>or</p> <pre>tspnet.write(id_instr, "*idn?\r\n")</pre>

upgrade functions

Use the functions in this section to perform upgrades.

upgrade.previous()	
Function	Upgrades a previous version of the Series 3700 firmware.
Usage	<code>upgrade.previous()</code>
Remarks	This command is equivalent to manually performing an upgrade to a previous version from the front panel. The functions will search the flash drive for a file to upgrade the unit and upgrade if one is found (if a file is not found, the function errors).

upgrade.unit()	
Function	Upgrades the Series 3700 firmware.
Usage	<code>upgrade.unit()</code>
Remarks	This command is equivalent to manually performing an upgrade from the front panel. The functions will search the flash drive for a file to upgrade the unit and upgrade if one is found (if a file is not found, the function errors).

userstring functions

Use the functions in this group to store/retrieve user-defined strings in nonvolatile memory.

userstring.add()	
Function	Adds a user-defined string to non- volatile memory.
Usage	<code>userstring.add(name, value)</code> 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 nonvolatile memory. The value associated with the given name can be retrieved with the userstring.get() (on page 13-311) function.
Also see	userstring.catalog() (on page 13-311) userstring.delete() (on page 13-311) userstring.get() (on page 13-311)
Example	Stores user-defined strings in non- volatile memory: <code>userstring.add("assetnumber", "236")</code> <code>userstring.add("department", "Widgets")</code> <code>userstring.add("contact", "John Doe")</code>

userstring.catalog()	
Function	Creates an iterator for the user string catalog.
Usage	<code>for name in userstring.catalog() do</code> <code>... end</code>
Remarks	Accessing the catalog for user string names allows the user to print or delete all string name values in nonvolatile memory. The entries will be enumerated in no particular order.
Also see	userstring.add() (on page 13-310) userstring.delete() (on page 13-311) userstring.get() (on page 13-311)

userstring.catalog()	
Example	<p>To delete all user strings in non- volatile memory:</p> <pre>for name in userstring.catalog() do userstring.delete(name) end</pre> <p>To print all user string name value pairs in nonvolatile memory:</p> <pre>for name in userstring.catalog() do print(name .. " = " .. userstring.get(name)) end</pre> <p>Output: department = Widgets assetnumber = 236 contact = John Doe</p> <p>The above output lists the user strings added in the "Example" for the userstring.add() (on page 13-310) function. Notice that they are not listed in the order that they were added.</p>

userstring.delete()	
Function	Deletes a user-defined string from nonvolatile memory.
Usage	<pre>userstring.delete(name)</pre> <p>name: Name of the user string.</p>
Remarks	This function will delete from non- volatile memory the string that is associated with the string name.
Also see	<p>userstring.add() (on page 13-310)</p> <p>userstring.catalog() (on page 13-310)</p> <p>userstring.get() (on page 13-311)</p>
Example	<p>Deletes user-defined strings from nonvolatile memory:</p> <pre>userstring.delete("assetnumber") userstring.delete("department") userstring.delete("contact")</pre>

userstring.get()	
Function	Retrieves a user-defined string from nonvolatile memory.
Usage	<pre>value = userstring.get(name)</pre> <p>name: Name of the user string. value: Returns the string value associated with name.</p>
Remarks	This function will retrieve from non- volatile memory the string that is associated with the string name.

userstring.get()	
Also see	userstring.add() (on page 13-310) userstring.catalog() (on page 13-310) userstring.delete() (on page 13-311)
Example	Retrieves the value for a user string from nonvolatile memory: <pre>value = userstring.get("assetnumber") print(value)</pre> Output: 236

waitcomplete functions

This function waits for all overlapped commands to complete.

waitcomplete()	
Function	Wait for all overlapped commands to complete.
Usage	<pre>waitcomplete([group])</pre> <p>group: Optional TSP-Link™ group on which to wait.</p>
Remarks	<p>This function will wait for all overlapped operations within given group to complete. A group number may only be specified from the master node. If no group is specified, the local group will be used. If zero is given for the group, this function will wait for all nodes in the system.</p> <hr/> <p>NOTE Any nodes that are not assigned to a group (their group number is zero) will be considered to be part of the master's group.</p> <hr/> <p>This function will wait for all previously started overlapped commands to complete. Currently the Series 3700 has no overlapped commands implemented. However, other TSP™-enabled products like the Series 3700 has overlapped commands. Therefore, when the Series 3700 is a TSP master to a slave device with overlapped commands, use this function to wait until all overlapped operations are completed.</p>

Verification

In this section:

Introduction	14-1
Verification test requirements.....	14-2
Performing the verification test procedures.....	14-5
Series 3700 verification tests	14-6

Introduction

Use the procedures in this section to verify that the Keithley Instruments Series 3700 System Switch/Multimeter's accuracy is within the limits stated in the instrument's one-year accuracy specifications. Verifying the accuracy of your Series 3700 is recommended:

- 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 that calibration 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, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

For the plug-in modules, the maximum common-mode voltage (voltage between any plug-in module terminal and chassis ground) is 300V DC or 300V RMS. Exceeding this value may cause a breakdown in insulation, creating a shock hazard.

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 these verification tests:

- Under the proper environmental conditions
- After the specified warmup period
- Using the correct line voltage
- Using the proper test equipment
- Using the specified output signal and reading limits

Environmental conditions

Conduct the verification procedures in a location that has:

- An ambient temperature of 18°C to 28°C (65°F to 82°F)
- A relative humidity of less than 80%, unless otherwise noted

Warmup period

NOTE At the factory, units are calibrated without any switch cards installed and all slots are covered with blank slot covers. The slot covers come installed on the unit when it is shipped.

If it is more convenient to calibrate the unit with switch cards installed, make sure all channels are open and any empty slots are covered with blank slot covers.

Allow the System Switch/Multimeter 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 in [Environmental conditions](#) (on page 14-2)), allow extra time for the instrument's internal temperature to stabilize. Typically, you need to 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 3700 requires a line voltage of 100V to 240V ($\pm 10\%$), and a line frequency of 50Hz or 60Hz.

NOTE The instrument automatically senses the line frequency at power-up.

Recommended test equipment

The following table summarizes recommended verification equipment. You can use alternate equipment if that equipment has specifications equal to or greater than those listed in the table. Note, however, that test equipment uncertainty will add to the uncertainty of each measurement. Generally, test equipment uncertainty should be at least four times better (more accurate) than corresponding Series 3700 specifications.

NOTE The Keithley Instruments Model 3706-190 backplane connector board is an accessory that can be used to make connections to the calibrator. Additional boards, such as a 4-wire short or the discrete resistors, would also be convenient to eliminate rewiring for different setups used in verification.

Manufacturer	Model	Description	Used for:	Uncertainty
Fluke	5700	Calibrator	All DCV, ACV, DCI, ACI, and resistance	See NOTE.
Fluke	5725	Amplifier	High voltage, high current	See NOTE.
HP	3458	DMM	10 μ A, 100 μ A DCI range	See NOTE.
Agilent	33220A	Function generator	Frequency	See NOTE.
N/A	N/A	4-wire short	DCV, resistance zeros	N/A
N/A	N/A	1 Ohm discrete resistor	1 Ohm range	+/- 20ppm
N/A	N/A	10 Ohm discrete resistor	10 Ohm range	+/- 20ppm

NOTE Refer to the manufacturer's specifications to calculate the uncertainty, which will vary for each test point.

Verification limits

The verification limits stated in this section have been calculated using only the Series 3700 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 3700 specifications and corresponding test equipment specifications.

Example reading limit calculation

The following is an example of how reading limits have been calculated. Assume you are testing the 10V DC range using a 10V input value. Using the Series 3700 one-year accuracy specification for 10V DC of \pm (25ppm of reading + 2ppm of range), the calculated limits are:

$$\text{Reading limits} = 10V \pm [(10V \times 25\text{ppm}) + (10V \times 2\text{ppm})]$$

$$\text{Reading limits} = 10V \pm (0.00025 + 0.00002)$$

$$\text{Reading limits} = 10V \pm 0.00027V$$

$$\text{Reading limits} = 9.99973V \text{ to } 10.00027V$$

Calculating resistance reading limits

Resistance reading limits must be recalculated based on the actual calibration resistance values supplied by the equipment manufacturer. Calculations are performed in the same manner as shown in the preceding example, using the actual calibration resistance values instead of the nominal values in the example when performing your calculations.

For example, assume that you are testing the 10k Ω range using an actual 10.03k Ω calibration resistance value. Using Series 3700 one-year 10k Ω range accuracy of \pm (60ppm of reading + 4ppm of range), the calculated reading limits are:

$$\text{Reading limits} = 10.03\text{k}\Omega \pm [(10.03\text{k}\Omega \times 60\text{ppm}) + (10.03\text{k}\Omega \times 4\text{ppm})]$$

$$\text{Reading limits} = 10.03\text{k}\Omega \pm [(0.000618) + (0.00004012)]$$

$$\text{Reading limits} = 10.03\text{k}\Omega \pm 0.0006798$$

$$\text{Reading limits} = 10.0293202\text{k}\Omega \text{ to } 10.0306798\text{k}\Omega$$

Restoring factory defaults

To restore the instrument to its factory front panel (bench) defaults before performing the verification procedures:

1. Press the **MENU** key.
2. Turn the navigation wheel to highlight **SETUP** and then press the **ENTER** key.
3. Turn the navigation wheel to highlight **RESET** and then press the **ENTER** key.

Performing the verification test procedures

The following topics provide a summary of verification test procedures, as well as items to take into consideration before performing any verification test.

Test summary

- [Verifying DC voltage](#) (on page 14-6)
- [Verifying AC voltage](#) (on page 14-9)
- [Verifying DC current 10 \$\mu\$ A to 100 \$\mu\$ A ranges](#) (on page 14-11)
- [Verifying DC current 1mA to 3A ranges](#) (on page 14-13)
- [Verifying AC current 1mA to 3A ranges](#) (on page 14-15)
- [Verifying frequency](#) (on page 14-18)
- [Verifying 4-wire resistance](#) (on page 14-19)
- [Verifying 2-wire resistance](#) (on page 14-21)
- [Verifying dry circuit resistance](#) (on page 14-22)
- [Verifying 1-OHM and 10-OHM resistance ranges](#) (on page 14-24)
- [Verifying zeros using a 4-wire short](#) (on page 14-25)

If the Series 3700 is not within specifications and not under warranty, calibrate the unit.

Test considerations

When performing the verification procedures:

- Be sure to restore factory front panel defaults as outlined in [Restoring factory defaults](#) (on page 14-5).
- Make sure that the test equipment is properly warmed up and connected to the Series 3700 terminals.
- Be sure the test equipment is set up for the proper function and range.
- Do not connect test equipment to the Series 3700 through a scanner, multiplexer, or other switching equipment.

WARNING *The input/output terminals of the digital multimeter (DMM) and switch cards are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500V peak. Do not connect the DMM or switch card terminals to CAT II, CAT III, or CAT IV circuits.*

Connections of the DMM or switch card terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltages.

Series 3700 verification tests

Perform these tests to verify the accuracy of your Series 3700 at the analog backplane connector.

Verifying DC voltage

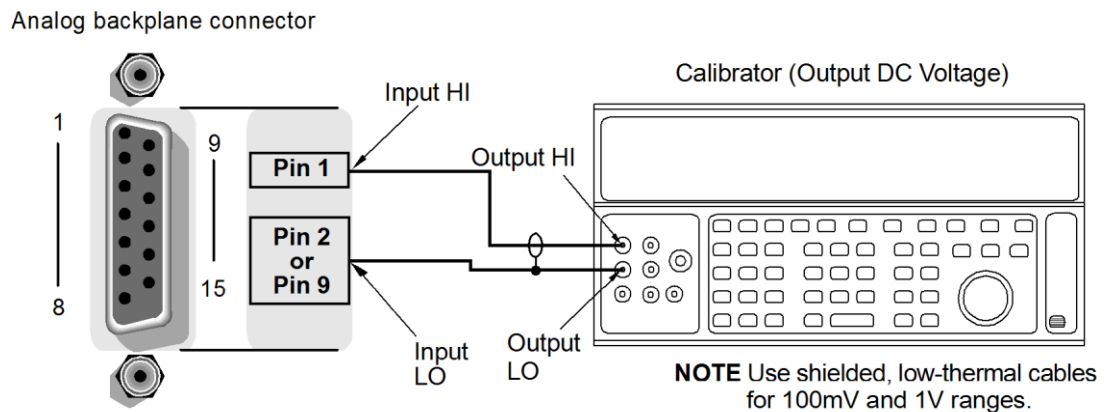
Check DC voltage accuracy by applying accurate voltages from the DC voltage calibrator to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

CAUTION Do not exceed 300V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify DC voltage accuracy:

NOTE Use shielded, low-thermal connections when testing the 100mV and 1V ranges to avoid errors caused by noise or thermal effects. Connect the shield to the calibrator's output LO terminal.

1. Connect the Series 3700 HI and LO INPUT pins to the DC voltage calibrator as shown in the "DC voltage verification" below.
2. Select the DC volts function.
3. Set the Series 3700 to the 100mV range.
4. If REL is needed, set the calibrator output to 0.00000mV DC and allow the reading to settle.
5. Enable the Series 3700 REL mode.
6. Source positive and negative full-scale and half-scale voltages for each of the ranges listed in the table below. For each voltage setting, be sure that the reading is within stated limits.

Figure 14-1: DC voltage verification**DC voltage verification data**

Use the following values to verify the performance of the Series 3700. Actual values depend on the published specifications (see [Example reading limit calculation](#) (on page 14-4)).

Connect to the Fluke 5700A Calibrator				
Description	Range (V)	Test point (V)	Lower limit (V)	Upper limit (V)
Rel Series 3700	1.00E-01	0.00E+00	N/A	N/A
Verify DCV 100mV	1.00E-01	1.00E-01	9.999610E-02	1.000039E-01

Connect to the Fluke 5700A Calibrator				
Description	Range (V)	Test point (V)	Lower limit (V)	Upper limit (V)
Verify DCV 100mV	1.00E-01	5.00E-02	4.999760E-02	5.000240E-02
Verify DCV 100mV	1.00E-01	-5.00E-02	-5.000240E-02	-4.999760E-02
Verify DCV 100mV	1.00E-01	-1.00E-01	-1.000039E-01	-9.999610E-02
Rel Series 3700	1.00E+00	0.00E+00	N/A	N/A
Verify DCV 1V	1.00E+00	1.00E+00	9.999680E-01	1.000032E+00
Verify DCV 1V	1.00E+00	5.00E-01	4.999830E-01	5.000170E-01
Verify DCV 1V	1.00E+00	-5.00E-01	-5.000170E-01	-4.999830E-01
Verify DCV 1V	1.00E+00	-1.00E+00	-1.000032E+00	-9.999680E-01
Verify DCV 10V	1.00E+01	1.00E+01	9.999730E+00	1.000027E+01
Verify DCV 10V	1.00E+01	5.00E+00	4.999855E+00	5.000145E+00
Verify DCV 10V	1.00E+01	0.00E+00	-2.000000E-05	2.000000E-05
Verify DCV 10V	1.00E+01	-5.00E+00	-5.000145E+00	-4.999855E+00
Verify DCV 10V	1.00E+01	-1.00E+01	-1.000027E+01	-9.999730E+00
Verify DCV 100V	1.00E+02	1.00E+02	9.999540E+01	1.000046E+02
Verify DCV 100V	1.00E+02	5.00E+01	4.999740E+01	5.000260E+01
Verify DCV 100V	1.00E+02	0.00E+00	-6.000000E-04	6.000000E-04
Verify DCV 100V	1.00E+02	-5.00E+01	-5.000260E+01	-4.999740E+01
Verify DCV 100V	1.00E+02	-1.00E+02	-1.000046E+02	-9.999540E+01

Connect to the Fluke 5700A Calibrator				
Description	Range (V)	Test point (V)	Lower limit (V)	Upper limit (V)
Verify DCV 300V	3.00E+02	3.00E+02	2.999862E+02	3.000138E+02
Verify DCV 300V	3.00E+02	1.50E+02	1.499922E+02	1.500078E+02
Verify DCV 300V	3.00E+02	0.00E+00	-1.800000E-03	1.800000E-03
Verify DCV 300V	3.00E+02	-1.50E+02	-1.500078E+02	-1.499922E+02
Verify DCV 300V	3.00E+02	-3.00E+02	-3.000138E+02	-2.999862E+02

Verifying AC voltage

Check AC voltage accuracy by applying accurate voltages from the AC voltage calibrator to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

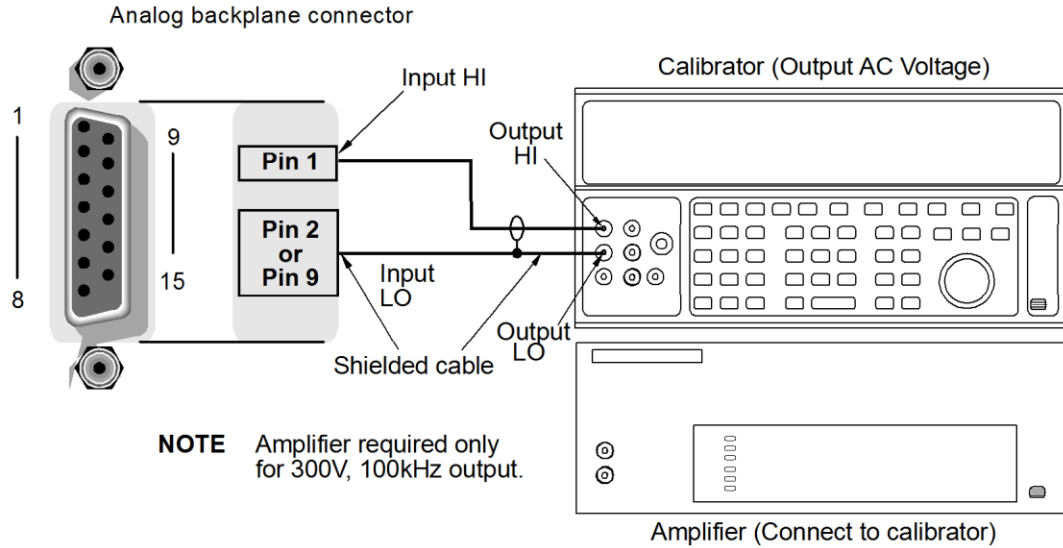
CAUTION Do not exceed 300V peak between INPUT HI and INPUT LO, or 8×10^7 VHz input, because instrument damage may occur.

To verify AC voltage accuracy:

NOTE Use shielded, low-thermal connections when testing the 100mV and 1V ranges to avoid errors caused by noise or thermal effects. Connect the shield to the calibrator's output LO terminal.

1. Connect the Series 3700 HI and LO INPUT pins to the DC voltage calibrator as shown in "AC voltage verification" below.
2. Select the AC volts function.
3. Set the Series 3700 to the 100mV range. Make sure that REL is disabled.
4. Source AC voltages for each of the frequencies and ranges are summarized in the [ACV verification data](#) (on page 14-10) table. For each setting, be sure that the reading is within stated limits.
5. Repeat steps 3 and 4 for each item in the table.

Figure 14-2: AC voltage verification



ACV verification data

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Example reading limit calculation](#) (on page 14-4)).

Connect to the Fluke 5700A calibrator				
Description	Range (V)	Test point (V)	Lower limit (V)	Upper limit (V)
Verify ACV 100mV @ 20Hz	1.00E-01	1.00E-01	9.897000E-02	1.010300E-01
Verify ACV 100mV @ 1kHz	1.00E-01	1.00E-01	9.992000E-02	1.000800E-01
Verify ACV 100mV @ 50kHz	1.00E-01	1.00E-01	9.984000E-02	1.001600E-01
Verify ACV 100mV @ 100kHz	1.00E-01	1.00E-01	9.932000E-02	1.006800E-01
Verify ACV 1V @ 20Hz	1.00E+00	1.00E+00	9.992000E-01	1.000800E+00
Verify ACV 1V @ 1kHz	1.00E+00	1.00E+00	9.992000E-01	1.000800E+00
Verify ACV 1V @ 50kHz	1.00E+00	1.00E+00	9.984000E-01	1.001600E+00
Verify ACV 1V @ 100kHz	1.00E+00	1.00E+00	9.932000E-01	1.006800E+00
Verify ACV 10V @ 1kHz	1.00E+01	1.00E+01	9.992000E+00	1.000800E+01
Verify ACV 10V @ 50kHz	1.00E+01	1.00E+01	9.984000E+00	1.001600E+01

Connect to the Fluke 5700A calibrator				
Description	Range (V)	Test point (V)	Lower limit (V)	Upper limit (V)
Verify ACV 10V @ 100kHz	1.00E+01	1.00E+01	9.932000E+00	1.006800E+01
Verify ACV 100V @ 1kHz	1.00E+02	1.00E+02	9.992000E+01	1.000800E+02
Verify ACV 100V @ 50kHz	1.00E+02	1.00E+02	9.984000E+01	1.001600E+02
Verify ACV 100V @ 100kHz	1.00E+02	1.00E+02	9.932000E+01	1.006800E+02
Verify ACV 300V @ 1kHz	3.00E+02	3.00E+02	2.997600E+02	3.002400E+02
Verify ACV 300V @ 50kHz	3.00E+02	3.00E+02	2.995200E+02	3.004800E+02
Connect to the Fluke 5725A amplifier				
Description	Range (V)	Test point (V)	Lower limit (V)	Upper limit (V)
Verify ACV 300V @ 100kHz	3.00E+02	3.00E+02	2.979600E+02	3.020400E+02

Verifying DC current 10 μ A to 100 μ A ranges

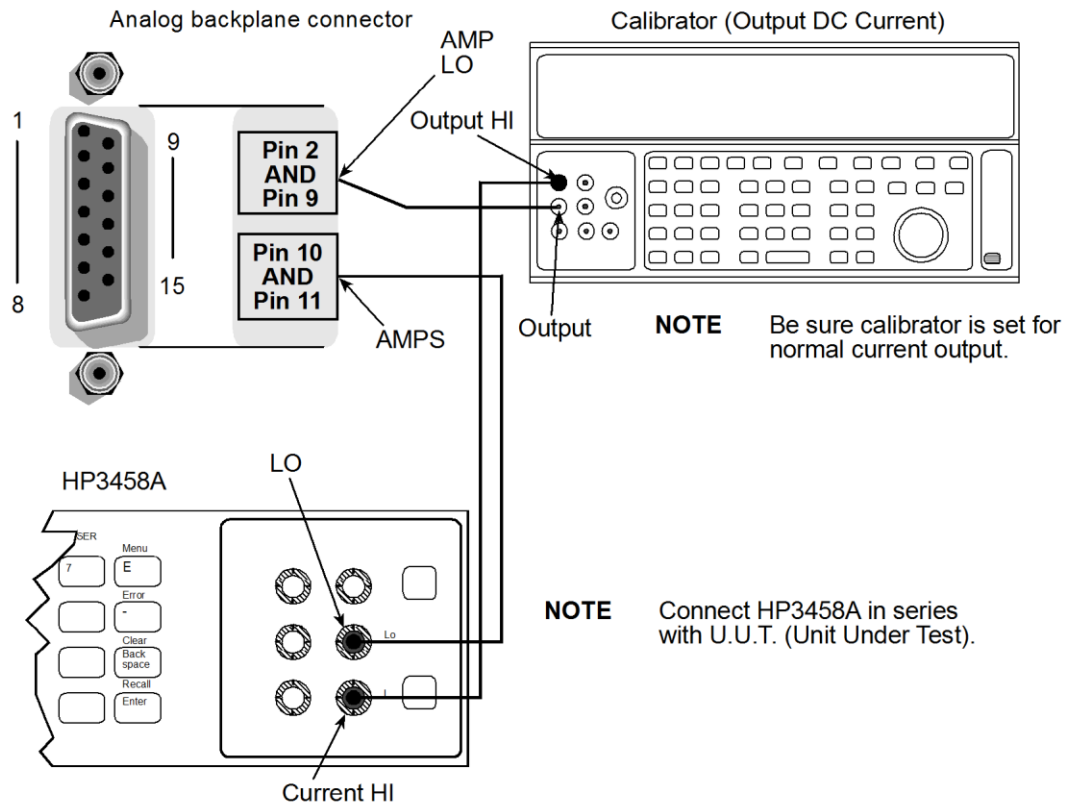
Check DC current accuracy by applying accurate current from the DC current calibrator to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

To verify DC current accuracy:

1. Set up the Series 3700 for DC current and the range being tested. Make sure REL is disabled.
2. Verify the zero test point for each range without any connection to the equipment and verify that the readings fall within specified limits.
3. Connect the Series 3700 AMPS and LO INPUT pins to the DC current calibrator as shown in the "DC current verification 10 μ A to 100 μ A ranges diagram" below.
4. Set up the HP3458A to the DC current function and range.

5. Set the calibrator to source zero current and rel both the Series 3700 and the HP3458A.
6. Source DC current for each of the test points summarized in the [DC voltage verification data](#) (on page 14-7) table. For each setting, be sure that the reading is within stated limits.

Figure 14-3: DC current verification 10µA to 100µA ranges



DC current verification data 10µA to 100µA ranges

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Example reading limit calculation](#) (on page 14-4)).

Connect HP3458A in series with 5700 calibrator				
Description	Range (A)	Test point(A)	Lower limit (A)	Upper limit (A)
Verify 10µA Zero	1.00E-05	0.00E+00	-3.000000E-10	3.000000E-10
Verify DC Curr 10µA	1.00E-05	1.00E-05	9.994700E-06	1.000530E-05
Verify DC Curr 10µA	1.00E-05	-1.00E-05	-1.000530E-05	-9.994700E-06

Connect HP3458A in series with 5700 calibrator				
Description	Range (A)	Test point(A)	Lower limit (A)	Upper limit (A)
Verify 100µA Zero	1.00E-04	0.00E+00	-3.000000E-09	3.000000E-09
Verify DC Curr 100µA	1.00E-04	1.00E-04	9.994910E-05	1.000509E-04
Verify DC Curr 100µA	1.00E-04	-1.00E-04	-1.000509E-04	-9.994910E-05

Verifying DC current 1mA to 3A ranges

Check DC current accuracy by applying accurate current from the DC current calibrator to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

NOTE The Fluke 5725A amplifier is only needed when verifying the 3A range.

To verify DC current accuracy:

1. Connect the Series 3700 AMPS and LO INPUT pins to the DC current calibrator as shown in the "DC current verification 1mA to 3A ranges diagram" below, using the Keithley Instruments Model 3706-751 fixture cable.
2. Select the DC current function.
3. Set the Series 3700 to the applicable ranges. Make sure that REL is disabled.
4. Source DC current for each of the test points summarized in the DC current verification data table. For each setting, be sure that the reading is within stated limits.

Figure 14-4: DC current verification 1mA to 3A ranges

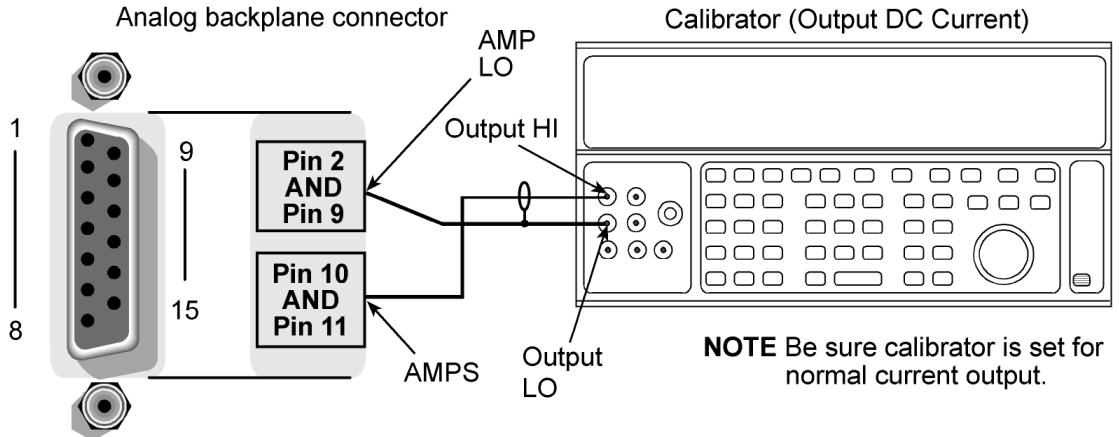
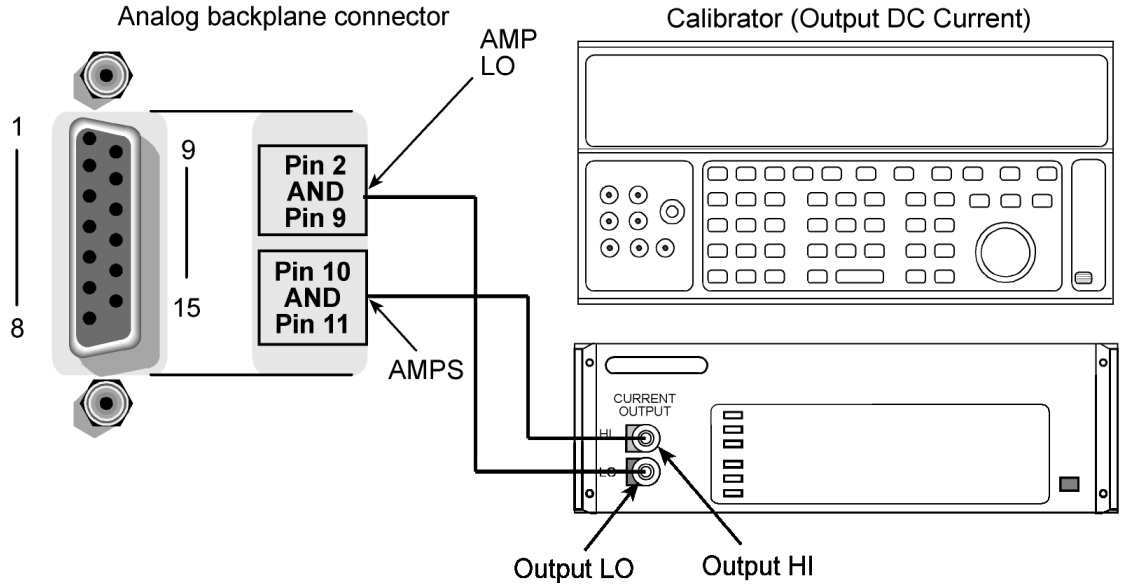


Figure 14-5: DC current verification 3A range diagram



DC current verification data 1mA to 3A ranges

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Example reading limit calculation](#) (on page 14-4)).

Remove HP3458A, only connect the 5700				
Description	Range (A)	Test point (A)	Lower limit (A)	Upper limit (A)
Verify 1mA Zero	1.00E-03	0.00E+00	-9.000000E-09	9.000000E-09
Verify DC Curr 1mA	1.00E-03	1.00E-03	9.994910E-04	1.000509E-03
Verify DC Curr 1mA	1.00E-03	-1.00E-03	-1.000509E-03	-9.994910E-04
Verify 10mA Zero	1.00E-02	0.00E+00	-9.000000E-08	9.000000E-08
Verify DC Curr 10mA	1.00E-02	1.00E-02	9.994910E-03	1.000509E-02
Verify DC Curr 10mA	1.00E-02	-1.00E-02	-1.000509E-02	-9.994910E-03
Verify 100mA Zero	1.00E-01	0.00E+00	-9.000000E-07	9.000000E-07
Verify DC Curr 100mA	1.00E-01	1.00E-01	9.994910E-02	1.000509E-01
Verify DC Curr 100mA	1.00E-01	-1.00E-01	-1.000509E-01	-9.994910E-02
Verify DC Curr 1A	1.00E+00	1.00E+00	9.991900E-01	1.000810E+00

Remove HP3458A, only connect the 5700				
Description	Range (A)	Test point (A)	Lower limit (A)	Upper limit (A)
Verify DC Curr 1A	1.00E+00	-1.00E+00	-1.000810E+00	-9.991900E-01
Connect to the Fluke 5725A amplifier				
Description	Range (A)	Test point (A)	Lower limit (A)	Upper limit (A)
Verify DC Curr 3A	3.00E+00	3.00E+00	2.996355E+00	3.003645E+00
Verify DC Curr 3A	3.00E+00	-3.00E+00	-3.003645E+00	- 2.996355E+00

Verifying AC current 1mA to 3A ranges

Check AC current accuracy by applying accurate current from the AC current calibrator at specific frequencies to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

To verify AC current accuracy:

1. Set up the Series 3700 for AC current and the range being tested. Make sure REL is disabled.
2. Source AC current for the 1mA to 1A range test points summarized in "AC current calibration diagram" below. For each setting, be sure that the reading is within stated limits.
3. Install the Fluke 5725A amplifier.
4. Source AC current for the 3A range test points summarized in the [AC current verification data 1mA to 1A ranges](#) (on page 14-16) table. Be sure that the 3A readings are within stated limits.

Figure 14-6: AC current verification 1mA to 1A range

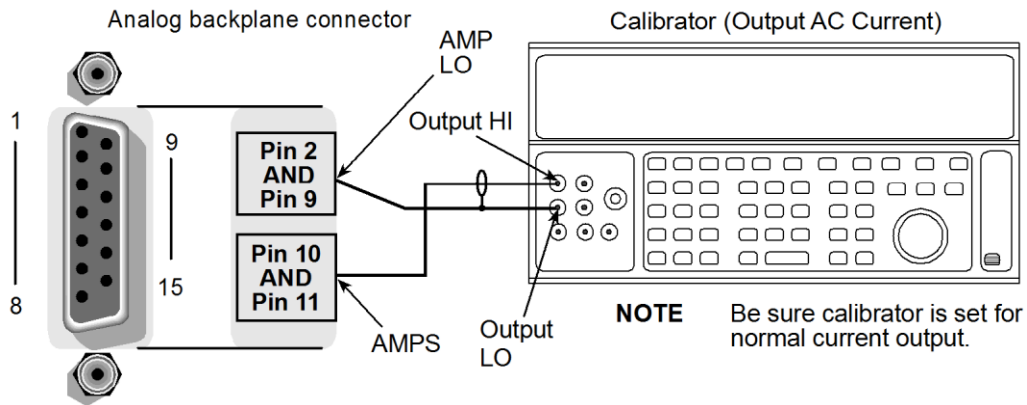
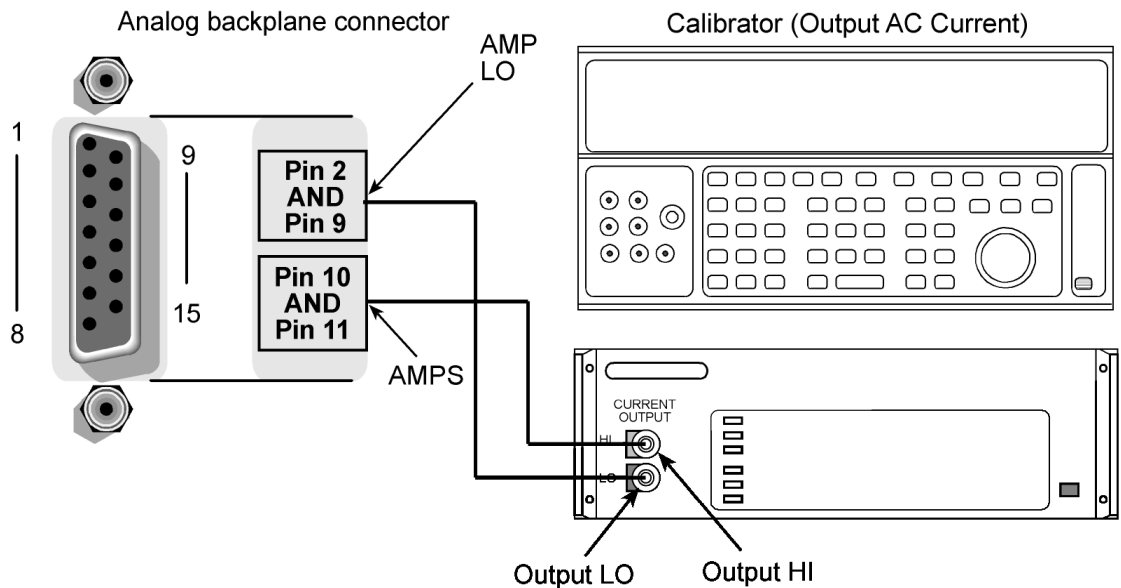


Figure 14-7: AC current verification 3A range



AC current verification data 1mA to 1A ranges

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Example reading limit calculation](#) (on page 14-4)).

Connect to the Fluke 5700A calibrator				
Description	Range (A)	Test point (A)	Lower limit (A)	Upper limit (A)
Verify AC Curr 1mA @ 20Hz	1.00E-03	1.00E-03	9.989000E-04	1.001100E-03

Connect to the Fluke 5700A calibrator				
Description	Range (A)	Test point (A)	Lower limit (A)	Upper limit (A)
Verify AC Curr 1mA @ 1kHz	1.00E-03	1.00E-03	9.989000E-04	1.001100E-03
Verify AC Curr 1mA @ 5kHz	1.00E-03	1.00E-03	9.989000E-04	1.001100E-03
Verify AC Curr 10mA @ 40Hz	1.00E-02	1.00E-02	9.989000E-03	1.001100E-02
Verify AC Curr 10mA @ 1kHz	1.00E-02	1.00E-02	9.989000E-03	1.001100E-02
Verify AC Curr 10mA @ 5kHz	1.00E-02	1.00E-02	9.989000E-03	1.001100E-02
Verify AC Curr 100mA @ 40Hz	1.00E-01	1.00E-01	9.989000E-02	1.001100E-01
Verify AC Curr 100mA @ 1kHz	1.00E-01	1.00E-01	9.989000E-02	1.001100E-01
Verify AC Curr 100mA @ 5kHz	1.00E-01	1.00E-01	9.989000E-02	1.001100E-01
Verify AC Curr 1A @ 40Hz	1.00E+00	1.00E+00	9.977000E-01	1.002300E+00
Verify AC Curr 1A @ 1kHz	1.00E+00	1.00E+00	9.977000E-01	1.002300E+00
Verify AC Curr 1A @ 5kHz	1.00E+00	1.00E+00	9.977000E-01	1.002300E+00

AC current verification data 3A range

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Example reading limit calculation](#) (on page 14-4)).

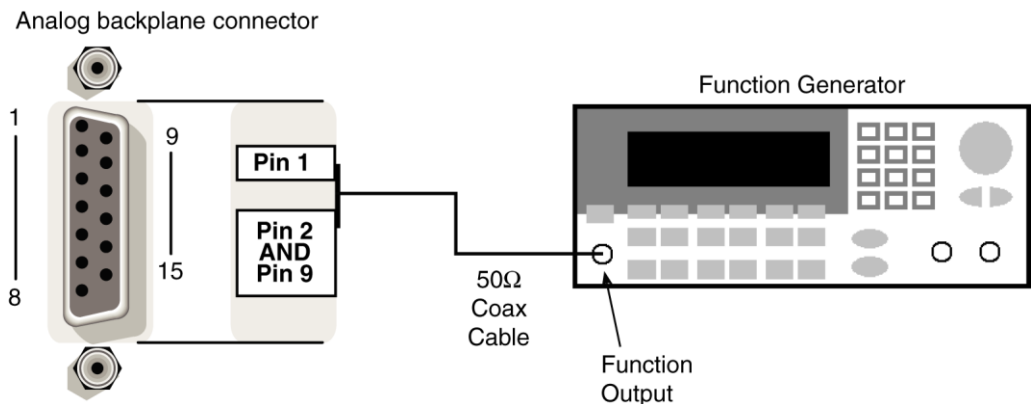
Connect to the Fluke 5725A amplifier				
Description	Range (A)	Test point (A)	Lower limit (A)	Upper limit (A)
Verify AC Curr 3A @ 40Hz	3.00E+00	3.00E+00	2.993100E+00	3.006900E+00
Verify AC Curr 3A @ 1kHz	3.00E+00	3.00E+00	2.993100E+00	3.006900E+00
Verify AC Curr 3A @ 5kHz	3.00E+00	3.00E+00	2.993100E+00	3.006900E+00

Verifying frequency

To verify the Series 3700 frequency function:

1. Connect the Agilent 33220A function generator to the Series 3700 INPUT pins.
2. Set the function generator to output a 1kHz, 5V RMS sine wave.
3. Select the Series 3700 frequency function by pressing the **FREQ** key.
4. Verify that each Series 3700 frequency reading is within the limits contained in the table contained in [Frequency verification data](#) (on page 14-18).

Figure 14-8: Frequency verification



Frequency verification data

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Example reading limit calculation](#) (on page 14-4)).

Connect the Agilent 33220A Generator				
Description	Range (V)	Frequency (Hz)	Lower limit (Hz)	Upper limit (Hz)
Verify Frequency 1kHz	1.00E+01	1.00E+03	9.999167E+02	1.000083E+03
Verify Frequency 10kHz	1.00E+01	1.00E+04	9.999167E+03	1.000083E+04
Verify Frequency 100kHz	1.00E+01	1.00E+05	9.999167E+04	1.000083E+05
Verify Frequency 250kHz	1.00E+01	2.50E+05	2.499797E+05	2.500203E+05
Verify Frequency 500kHz	1.00E+01	5.00E+05	4.999597E+05	5.000403E+05

Verifying 4-wire resistance

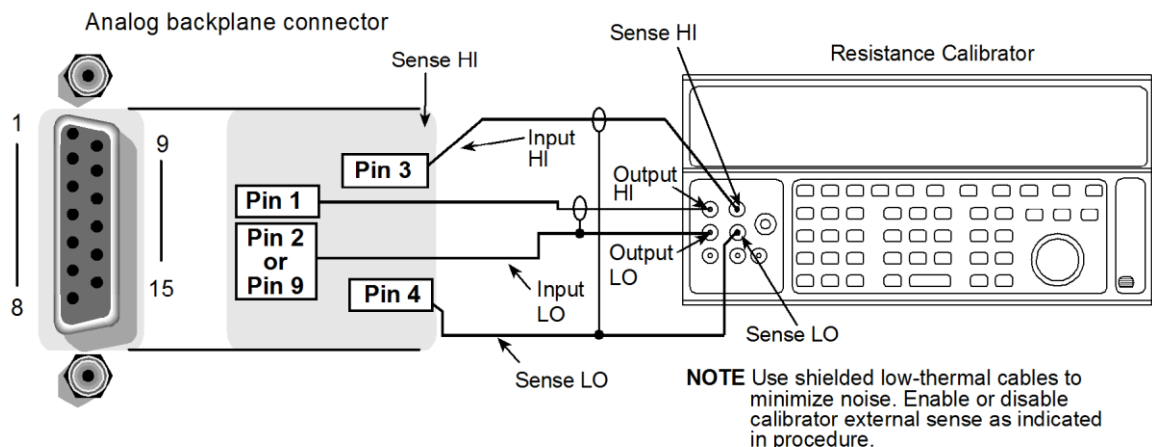
Check the normal resistance function by connecting accurate resistance values to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

CAUTION Do not exceed 300V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify 4-wire resistance accuracy:

1. Using shielded, Teflon-insulated or equivalent cables in a 4-wire configuration, connect the Series 3700 INPUT and SENSE pins to the calibrator as shown for 100 Ω to 10M Ω ranges.
2. Set the calibrator for 4-wire resistance with external sense on.
3. Select the Series 3700 4-wire resistance function.
4. Select the SLOW integration rate with the **RATE** key.
5. Set the Series 3700 for the 100 Ω range, and make sure the FILTER is on. Enable OC+ (offset-compensated ohms). Use OC+ for 100 Ω and 1k Ω range verification only. See Enabling/disabling offset-compensated ohms in the User's manual.
6. Recalculate reading limits based on actual calibrator resistance values.
7. Source the nominal full-scale resistance values for the 100 Ω to 10M Ω ranges summarized in the [4-wire resistance verification data](#) (on page 14-20) table. Recalculate the limits based on the actual value of the resistor and verify the reading is within the calculated limits.

Figure 14-9: Resistance verification



4-wire resistance verification data

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Calculating resistance reading limits](#) (on page 14-4)).

Connect to the Fluke 5700A calibrator				
Description	Range (Ohms)	Test point (Ohms)	Lower limit (Ohms)	Upper limit (Ohms)
Verify 4W Res 100 Ohm *	1.00E+02	1.00E+02	9.999310E+01	1.000069E+02
Verify 4W Res 1k Ohm	1.00E+03	1.00E+03	9.999360E+02	1.000064E+03
Verify 4W Res 10k Ohm	1.00E+04	1.00E+04	9.999360E+03	1.000064E+04
Verify 4W Res 100k Ohm	1.00E+05	1.00E+05	9.999360E+04	1.000064E+05
Verify 4W Res 1M Ohm	1.00E+06	1.00E+06	9.999560E+05	1.000044E+06
Verify 4W Res 10M Ohm	1.00E+07	1.00E+07	9.995900E+06	1.000410E+07

NOTE The asterisk (*) designates the ranges that offset compensation is being used.

Verifying 2-wire resistance

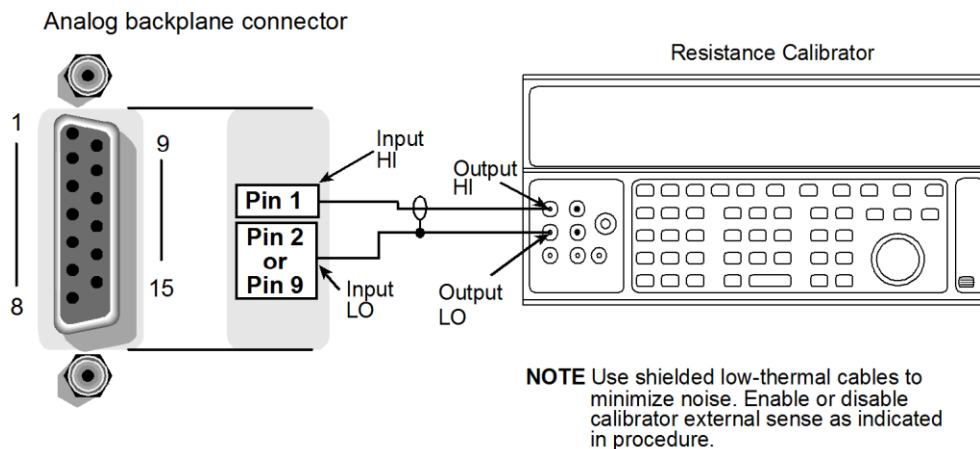
Check the normal resistance function by connecting accurate resistance values to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

CAUTION Do not exceed 300V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify normal resistance accuracy:

1. Using shielded, Teflon-insulated or equivalent cables in a 2-wire configuration, connect the Series 3700 INPUT and SENSE pins to the calibrator as shown in the "2-wire resistance verification diagram" below.
2. Disable the external sense on the calibrator.
3. Set the Series 3700 to the 2-wire resistance function, set to the proper range.
4. Source a nominal 100k Ω -100M Ω resistance value. Recalculate the limits based on the actual value of the resistor and verify that the reading is within the calculated limits.

Figure 14-10: 2-wire resistance verification



2-wire resistance verification data

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Calculating resistance reading limits](#) (on page 14-4)).

Description	Range (Ohms)	Test point (Ohms)	Lower limit (Ohms)	Upper limit (Ohms)
Verify 2W Res 100k Ohm	1.00E+05	1.00E+05	9.999360E+04	1.000064E+05
Verify 2W Res 1M Ohm	1.00E+06	1.00E+06	9.999360E+05	1.000064E+06
Verify 2W Res 10M Ohm	1.00E+07	1.00E+07	9.995900E+06	1.000410E+07
Verify 2W Res 100M Ohm	1.00E+08	1.00E+08	9.979700E+07	1.002030E+08

Verifying dry circuit resistance

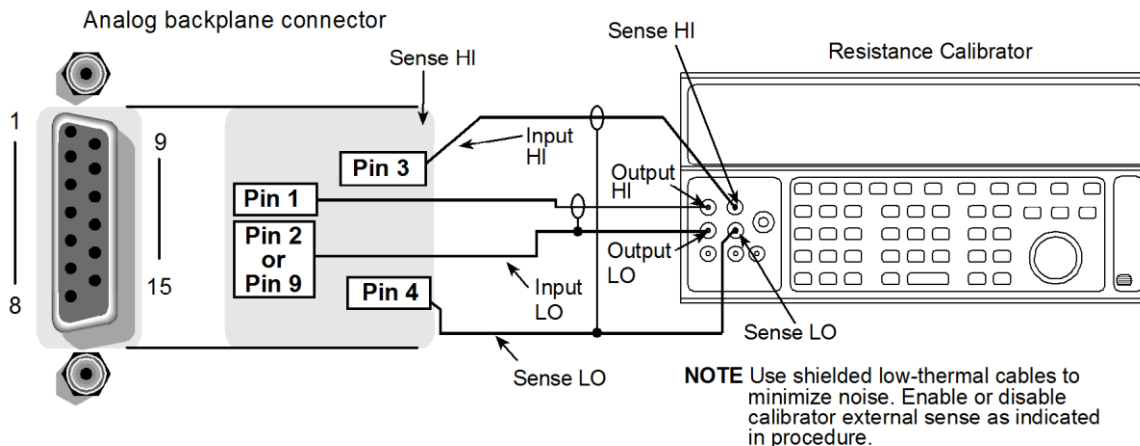
Check the dry circuit resistance function by connecting accurate resistance values to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

CAUTION Do not exceed 300V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify dry circuit resistance accuracy:

- Using shielded, Teflon-insulated or equivalent cables in a 4-wire configuration, connect the Series 3700 INPUT and SENSE pins to the calibrator as shown for 100Ω to 10MΩ ranges.
- Set the calibrator for 4-wire resistance with external sense on.
- Select the Series 3700 4-wire resistance function.
- Select the SLOW integration rate with the **RATE** key.
- Enable dry circuit resistance function (see Enabling/disabling dry circuit ohms in the User's manual).
- Set the Series 3700 for the 100Ω range, and make sure the FILTER is on. Enable OC+ (offset-compensated ohms). Use OC+ for 100Ω, and 1kOhm range verification. See Enabling/disabling offset-compensated ohms in the User's manual.
- Recalculate reading limits based on actual calibrator resistance values.
- Source the nominal full-scale resistance values for the 100Ω to 2kΩ ranges summarized in the [Dry circuit resistance verification data](#) (on page 14-23) table. Verify that the readings are within calculated limits.

Figure 14-11: Resistance verification



Dry circuit resistance verification data

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Calculating resistance reading limits](#) (on page 14-4)).

Description	Range (Ohms)	Test point (Ohms)	Lower limit (Ohms)	Upper limit (Ohms)
Verify Dry Circuit 100 Ohm *	1.00E+02	1.00E+02	9.997800E+01	1.000220E+02
Verify Dry Circuit 1k Ohm	1.00E+03	1.00E+03	9.995200E+02	1.000480E+03
Verify Dry Circuit 2k Ohm	2.00E+03	1.90E+03	1.898320E+03	1.901680E+03

NOTE The asterisk (*) designates the ranges that offset compensation is being used.

Verifying 1-OHM and 10-OHM resistance ranges

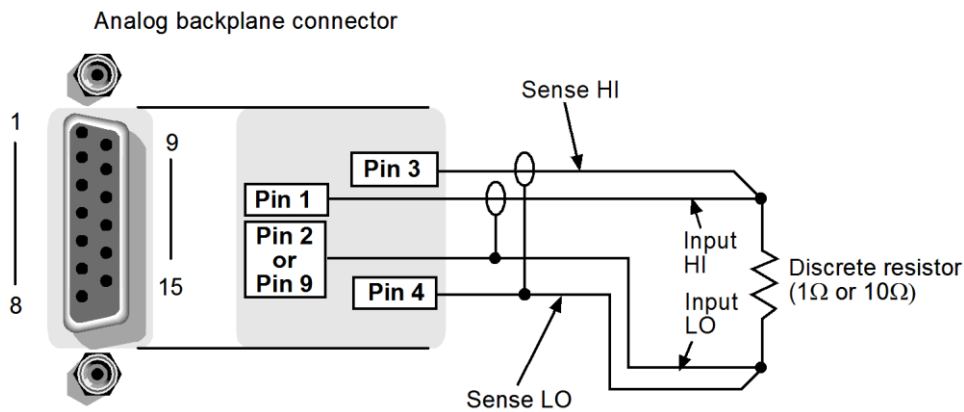
Check the normal resistance function by connecting accurate resistance values to the Series 3700 analog backplane connector and verifying that the displayed readings fall within specified limits.

CAUTION Do not exceed 300V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify normal resistance accuracy:

1. Connect the 1Ω discrete resistor to the Series 3700 input.
2. For the dry circuit test points, enable the dry circuit resistance attribute (DRY+).
3. Select the SLOW integration rate with the **RATE** key.
4. Set the Series 3700 for the 1Ω range, and make sure the FILTER is on. Enable OC+ (offset-compensated ohms). Use OC+ for 1Ω and 10Ω range verification.
5. Recalculate reading limits based on actual discrete resistor resistance values.
6. Repeat using the 10Ω discrete resistor on the 10Ω range.

Figure 14-12: Verifying discrete resistance



Discrete resistance verification data

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Calculating resistance reading limits](#) (on page 14-4)).

1 Ohm discrete resistor applied				
Description	Range (Ohms)	Test point (Ohms)	Lower limit (Ohms)	Upper limit (Ohms)
Verify Res 1 Ohm *	1.00E+00	1.00E+00	9.998600E-01	1.000140E+00
Verify Dry Circuit 1 Ohm *	1.00E+00	1.00E+00	9.998500E-01	1.000150E+00

10 Ohm discrete resistor applied				
Description	Range (Ohms)	Test point (Ohms)	Lower limit (Ohms)	Upper limit (Ohms)
Verify Res 10 Ohm *	1.00E+01	1.00E+01	9.999310E+00	1.000069E+01
Verify Dry Circuit 10 Ohm *	1.00E+01	1.00E+01	9.998500E+00	1.000150E+01

NOTE The asterisk (*) designates the ranges that offset compensation is being used.

Verifying zeros using a 4-wire short

Check the zeros of various test points while the 4-wire is connected to the Series 3700 analog backplane connector and verify that the displayed readings fall within specified limits.

CAUTION Do not exceed 300V peak between INPUT HI and INPUT LO because instrument damage may occur.

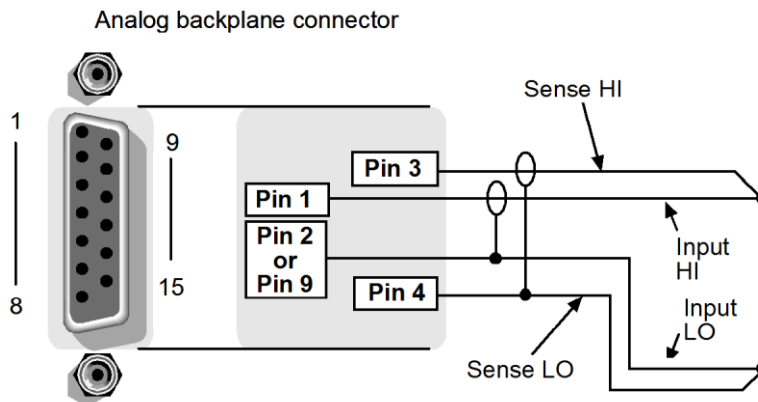
To verify DC voltage and resistance zeros:

1. Select the DC volts function.
2. Set the Series 3700 to the 100mV range.
3. Connect the 4-wire short to the Series 3700 analog backplane connector and allow to settle for 5 minutes (do not use REL).
4. Verify the 100mV zero is within specification (see the [4-wire short applied verification data](#) (on page 14-26) table).
5. Set the Series 3700 to the 1V range.
6. Allow to settle for 30 seconds (do not use REL).
7. Verify the 1V zero is within specification (see the [4-wire short applied verification data](#) (on page 14-26) table).

To verify resistance using the 4-wire short:

1. With the 4-wire short still applied, select the Series 3700 4-wire resistance function.
2. Select the SLOW integration rate with the **RATE** key.
3. Set the Series 3700 for the 1Ω range, and make sure the FILTER is on. Enable OC+ (offset-compensated ohms). Use OC+ for 1Ω and 10Ω range verification.
4. Verify the 1Ω range zero is within specification (see the [4-wire short applied verification data](#) (on page 14-26) table).
5. Set the Series 3700 for the 10Ω range (make sure the FILTER is on and OC+ is still enabled).
6. Verify the 10Ω range zero is within specification (see the [4-wire short applied verification data](#) (on page 14-26) table).

Figure 14-13: 4-wire short diagram



4-wire short applied verification data

Use the following values to verify the performance of the Series 3700. Actual values depend on published specifications (see [Calculating resistance reading limits](#) (on page 14-4)).

4-wire short applied				
Description	Range (V)	Test point (V)	Lower limit (V)	Upper limit (V)
Verify Zeros 100mV DC	1.00E-01	0.00E+00	-9.000000E-07	9.000000E-07
Verify Zeros 1V DC	1.00E+00	0.00E+00	-2.000000E-06	2.000000E-06

Description	Range (Ohms)	Test point (Ohms)	Lower limit (Ohms)	Upper limit (Ohms)
Verify Zeros 1 Ohm *	1.00E+00	0.00E+00	-8.000000E-05	8.000000E-05
Verify Zeros 10 Ohm *	1.00E+01	0.00E+00	-9.000000E-05	9.000000E-05

NOTE The asterisk (*) designates the ranges that offset compensation is being used.

This completes the verification procedure.

In this section:

Overview	15-1
Environmental conditions	15-2
Calibration considerations	15-3
Calibration	15-4
Remote calibration procedure	15-5

Overview

Use the procedures in this section to calibrate the Keithley Instruments Series 3700 System Switch/Multimeter.

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, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

For the plug-in modules, the maximum common-mode voltage (voltage between any plug-in module terminal and chassis ground) is 300V DC or 300V RMS. Exceeding this value may cause a breakdown in insulation, creating a shock hazard.

All procedures in this section require accurate calibration equipment to supply precise DC and AC voltages, DC and AC currents, and resistance values. Calibration can be performed any time by an operator using the Instrument Control Language (ICL) commands sent either over the IEEE-488 bus or Ethernet. DC-only or AC-only calibration may be performed individually, if desired.

Environmental conditions

Conduct the verification procedures in a location that has:

- An ambient temperature of 18°C to 28°C (65°F to 82°F)
- A relative humidity of less than 80%, unless otherwise noted

Warmup period

NOTE At the factory, units are calibrated without any switch cards installed and all slots are covered with blank slot covers. The slot covers come installed on the unit when it is shipped.

If it is more convenient to calibrate the unit with switch cards installed, make sure all channels are open and any empty slots are covered with blank slot covers.

Allow the System Switch/Multimeter 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 in [Environmental conditions](#) (on page 14-2)), allow extra time for the instrument's internal temperature to stabilize. Typically, you need to 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 3700 requires a line voltage of 100V to 240V ($\pm 10\%$), and a line frequency of 50Hz or 60Hz.

NOTE The instrument automatically senses the line frequency at power-up.

Calibration considerations

When performing the calibration procedures:

- Make sure that the equipment is properly warmed up and connected to the appropriate input jacks.
- Make sure the calibrator is in OPERATE mode before you complete each calibration step.
- Always let the source signal settle before calibrating each point.
- If an error occurs during calibration, the Series 3700 will generate an appropriate error message. See [Error summary](#) (on page 17-1) for more information.

WARNING *The input/output terminals of the digital multimeter (DMM) and switch cards are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500V peak. Do not connect the DMM or switch card terminals to CAT II, CAT III, or CAT IV circuits.*

Connections of the DMM or switch card terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltages.

Calibration cycle

Perform calibration at least once a year, or every 90 days to ensure the unit meets the corresponding specifications.

Recommended equipment

The following table lists the recommended equipment and settings you need for DC-only, and AC-only calibration procedures. Alternate equipment may be used, such as a DC transfer standard and characterized resistors, as long as the equipment has specifications at least as good as those listed in the table. In general, equipment uncertainty should be at least four times better (more accurate) than the corresponding Series 3700 specifications.

NOTE The Keithley Instruments Model 3706-190 backplane connector board is an accessory that can be used to make connections to the calibrator. Additional boards, such as a 4-wire short or the discrete resistors, would also be convenient to eliminate rewiring for different setups used in verification.

Manufacturer	Model	Description	Used for:	Uncertainty
Fluke	5700	Calibrator	All DCV, ACV, DCI, ACI, and Resistance	See NOTE.
N/A	N/A	4-wire short	DCV, resistance zeros	N/A
Agilent	33220 A	Function generator	For frequency factory cal only	See NOTE.

NOTE Refer to the manufacturer's specifications to calculate the uncertainty, which will vary for each test point.

Calibration

Calibration must be performed by remote control using Ethernet, GPIB, or USB interfaces. No front panel calibration is available. Refer to [System connections](#) (on page 2-10) for more information on communicating with the instrument.

"Factory calibration" refers to additional calibration steps that are only performed once at the factory or when a unit has been repaired by replacing PC boards or components of the boards. The remaining calibration steps can be performed as needed.

The factory calibration steps are:

- **DC Cal Step 0:** A/D MUX Offset, which is performed at the beginning prior to other DC calibration steps
- **Frequency Cal step 17:** 1V @ 10Hz and step 18: 1V @ 1kHz, which are performed at the end of AC calibration

You can perform individual sections of calibration, but for the instrument to be calibrated properly, all the steps of a section should be performed. For example, DC Cal Step 1: 4-wire short should be done as well as Steps 2 through 5 to properly calibrate DC volts. Other sections are resistance, DC current, AC volts, and AC current. Calibration must be saved at the end in order for the adjustment to be permanent.

Before performing a calibration, check the system date of the Series 3700. This can be done by sending the following command:

```
print(os.date("%x"))
```

If the date is wrong, the date and time need to be reset using the following command:

```
settime(os.time(year = yyyy, month = mm, day = dd, hour = hh, min = mm, sec = ss))
```

Make sure to enter the correct date and time using the 24-hour clock. If the date is incorrect, it will not save the proper date when calibration is saved. For additional information about this command, see `localnode.settime()` (on page 13-215).

Remote calibration procedure

To perform calibration, use the following procedure:

1. Connect the Series 3700 to the IEEE-488 bus of the computer using a shielded IEEE-488 cable, such as the Keithley Instruments Model 7007, over the Ethernet, or directly to a computer through the Ethernet port using a cross-over cable.
2. Turn on the Series 3700 and allow it to warm up for at least two hours before performing calibration.
3. Make sure the primary address of the Series 3700 is the same as the address specified in the program that you will be using to send commands (the GPIB default address is 16; the Ethernet default port number is 23).
4. Turn the TSP™ prompt and errors off and unlock the calibration function by sending the following commands:

```
SEND localnode.prompts=0
SEND localnode.showerrors=0
SEND dmm.reset()
SEND errorqueue.clear()
SEND dmm.calibration.unlock("KI003706")
```

NOTE When remotely changing the unlock code, send the `dmm.calibration.unlock()` (on page 13-122) command twice, first with the present code, then with the new code.

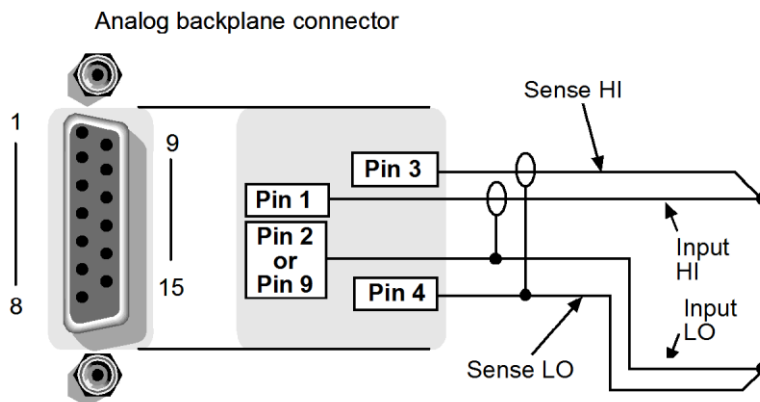
5. Check for errors after sending each calibration command by using the following command:

```
SEND print(errorqueue.count)
```
6. Send each calibration command with `print ("done")` appended to allow the program to know when operation is complete. Some calibration steps may take up to five minutes to perform, so the communication time-out setting should be adjusted, because otherwise time-out errors might occur.

DC volts calibration

1. Install the 4-wire short on the analog backplane connector inputs of the Series 3700.
2. Allow the unit to settle for five minutes.
3. Perform the following calibration steps (DC Cal Step 0 through Step 5):

Figure 15-1: 4-wire short diagram



DC Cal Step 0: A/D MUX Offset Cal (factory cal only)

Send the following commands:

```
SEND dmm.calibration.dc(0) print("done")
SEND print(errorqueue.count)
```

DC Cal Step 1: Input short circuit

1. Allow the unit to settle for 30 seconds.
2. Send the following commands:

```
SEND dmm.calibration.dc(1) print("done")
SEND print(errorqueue.count)
```

DC Cal Step 2: Open input

1. Remove the 4-wire short from the inputs.

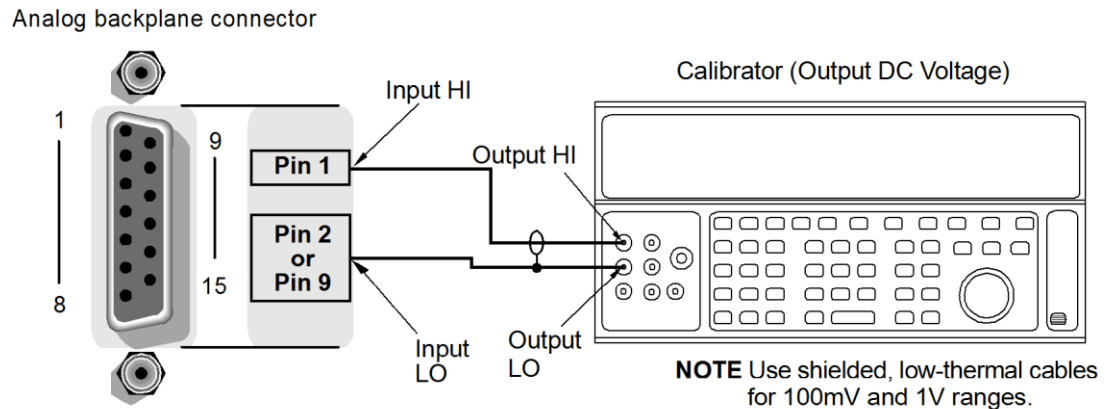
NOTE DO NOT install cables to the inputs (cables will be installed in [DC Cal Step 3: +10 Volt](#) (on page 15-7)).

2. Send the following commands:

```
SEND dmm.calibration.dc(2) print("done")
SEND print(errorqueue.count)
```

DC Cal Step 3: +10 Volt

Figure 15-2: DC voltage calibration



1. Connect a cable between the calibrator and the Series 3700.
2. Allow the unit to settle for 30 seconds.
3. Send the following command:

```
SEND dmm.range = 10
```

4. Source +10V.
5. Send the following commands:

```
SEND dmm.calibration.dc(3,10) print("done")
SEND print(errorqueue.count)
```

DC Cal Step 4: -10 Volt

1. Source -10V.
2. Send the following commands:

```
SEND dmm.calibration.dc(4,-10) print("done")
SEND print(errorqueue.count)
```

DC Cal Step 5: 100 Volt

1. Send the following command:

```
SEND dmm.range = 100
```

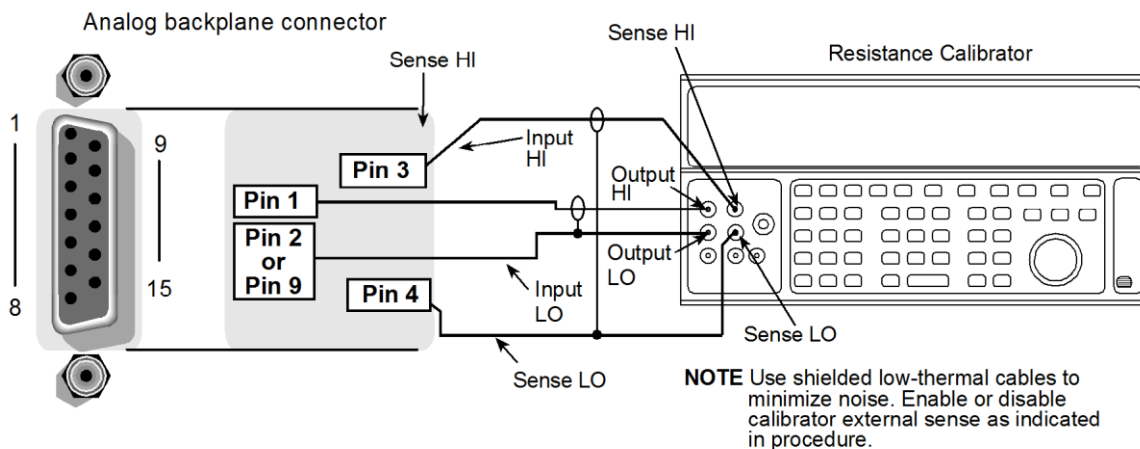
2. Source 100V.
3. Send the following commands:

```
SEND dmm.calibration.dc(5,100) print("done")
SEND print(errorqueue.count)
```

Resistance calibration

Perform the following calibration steps (DC Cal Step 6 through Step 9):

Figure 15-3: Resistance calibration



DC Cal Step 6: 100 Ohm

1. Send the following commands:

```
SEND dmm.func = dmm.four_wire_ohms
SEND dmm.range = 100
```

2. Source 100 Ohms, and then read the resistor value from the calibrator.

3. Send the following command:

```
SEND dmm.calibration.dc(6, (resistor value))
print("done")
```

DC Cal Step 7: 10k Ohm

1. Send the following command:

```
SEND dmm.range = 10e+3
```

2. Source 10k Ohm, and then read the resistor value from the calibrator.

3. Send the following command:

```
SEND dmm.calibration.dc(7, (resistor value))
print("done")
```

DC Cal Step 8: 100k Ohm

1. Send the following command:

```
SEND dmm.range = 100e+3
```
2. Source 100k Ohm, and then read the resistor value from the calibrator.
3. Send the following command:

```
SEND dmm.calibration.dc(8, (resistor value))
print("done")
```

DC Cal Step 9: 1M Ohm

1. Send the following command:

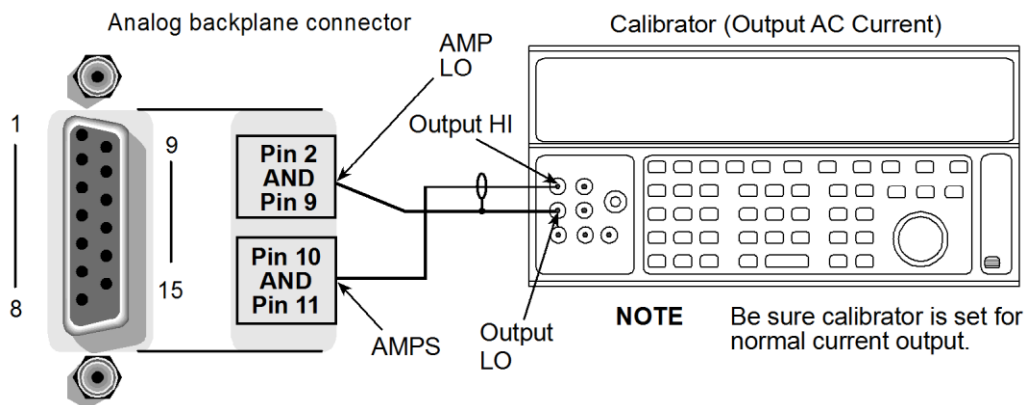
```
SEND dmm.range = 1e+6
```
2. Source 1M Ohm then read the resistor value from the calibrator.
3. Send the following command:

```
SEND dmm.calibration.dc(9, (resistor value))
print("done")
```

DC current calibration

Make the connections as shown, then perform the following calibration steps (DC Cal Step 10 through Step 14):

Figure 15-4: DC current calibration



DC Cal Step 10: 100 μ A

1. Send the following commands:

```
SEND dmm.func = dmm.dc_current  
SEND dmm.range = 100e-6
```

2. Source 100 μ A.

3. Send the following commands:

```
SEND dmm.calibration.dc(10,.0001) print("done")
```

DC Cal Step 11: 1mA

1. Send the following command:

```
SEND dmm.range = 1e-3
```

2. Source 1mA.

3. Send the following command:

```
SEND dmm.calibration.dc(11,.001) print("done")
```

DC Cal Step 12: 10mA

1. Send the following command:

```
SEND dmm.range = 10e-3
```

2. Source 10mA.

3. Send the following command:

```
SEND dmm.calibration.dc(12,.01) print("done")
```

DC Cal Step 13: 100mA

1. Send the following command:

```
SEND dmm.range = 100e-3
```

2. Source 100mA.

3. Send the following command:

```
SEND dmm.calibration.dc(13,.1) print("done")
```

DC Cal Step 14: 1A

1. Send the following command:

```
SEND dmm.range = 1
```

2. Source 1A.

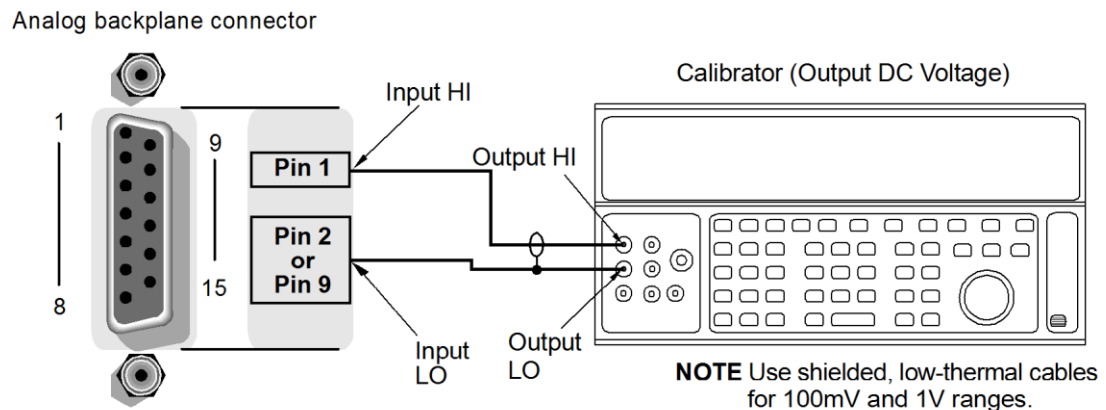
3. Send the following command:

```
SEND dmm.calibration.dc(14,1) print("done")
```


AC volts calibration

Make the connections as shown below, then perform the following calibration steps (AC Cal Step 1 through Step 10):

Figure 15-5: AC voltage calibration



AC Cal Step 1: 10mV @ 1kHz

1. Send the following commands:


```
SEND dmm.func = dmm.ac_volts
SEND dmm.range = 10e-3
```
2. Source 10mV @ 1kHz.
3. Send the following command:


```
SEND dmm.calibration.ac(1) print("done")
```

AC Cal Step 2: 100mV @ 1kHz

1. Send the following command:


```
SEND dmm.range = 100e-3
```
2. Source 100mV @ 1kHz.
3. Send the following command:


```
SEND dmm.calibration.ac(2) print("done")
```

AC Cal Step 3: 100mV @ 50kHz

1. Source 100mV @ 50kHz.
2. Send the following command:


```
SEND dmm.calibration.ac(3) print("done")
```

AC Cal Step 4: 1V @ 1kHz

1. Send the following command:

```
SEND dmm.range = 1
```

2. Source 1V @ 1kHz.

3. Send the following command:

```
SEND dmm.calibration.ac(4) print("done")
```

AC Cal Step 5: 1V @ 50kHz

1. Source 1V @ 50kHz.

2. Send the following command:

```
SEND dmm.calibration.ac(5) print("done")
```

AC Cal Step 6: 10V @ 1kHz

1. Send the following command:

```
SEND dmm.range = 10
```

2. Source 10V @ 1kHz.

3. Send the following command:

```
SEND dmm.calibration.ac(6) print("done")
```

AC Cal Step 7: 10V @ 50kHz

1. Source 10V @ 50kHz.

2. Send the following command:

```
SEND dmm.calibration.ac(7) print("done")
```

AC Cal Step 8: 100V @ 1kHz

1. Send the following command:

```
SEND dmm.range = 100
```

2. Source 100V @ 1kHz.

3. Send the following command:

```
SEND dmm.calibration.ac(8) print("done")
```

AC Cal Step 9: 100V @ 50kHz

1. Source 100V @ 50kHz.

2. Send the following command:

```
SEND dmm.calibration.ac(9) print("done")
```

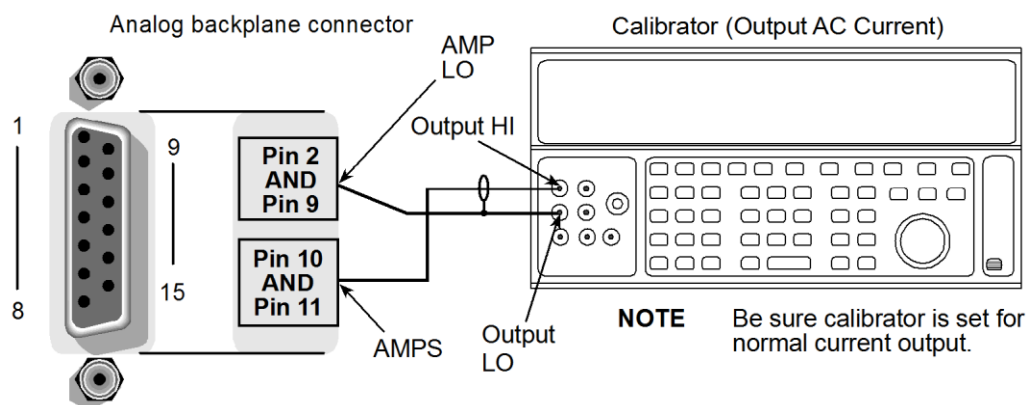
AC Cal Step 10: 300V @ 1kHz

1. Send the following command:
`SEND dmm.range = 300`
2. Source 300V @ 1kHz
3. Send the following command:
`SEND dmm.calibration.ac(10) print("done")`

AC current calibration

Make the connections as shown, then perform the following calibration steps (AC Cal Step 11 through Step 16):

Figure 15-6: AC current calibration 1mA to 1A range



AC Cal Step 11: 100 μ A @ 1kHz

1. Send the following commands:
`SEND dmm.func = dmm.ac_current`
`SEND dmm.range = 100e-6`
2. Source 100 μ A @ 1kHz.
3. Send the following command:
`SEND dmm.calibration.ac(11) print("done")`

AC Cal Step 12: 1mA @ 1kHz

1. Send the following command:
`SEND dmm.range = 1e-3`
2. Source 1mA @ 1kHz.
3. Send the following command:
`SEND dmm.calibration.ac(12) print("done")`

AC Cal Step 13: 10mA @ 1kHz

1. Send the following command:
`SEND dmm.range = 10e-3`
2. Source 10mA @ 1kHz.
3. Send the following command:
`SEND dmm.calibration.ac(13) print("done")`

AC Cal Step 14: 100mA @ 1kHz

1. Send the following command:
`SEND dmm.range = 100e-3`
2. Source 100mA @ 1kHz.
3. Send the following command:
`SEND dmm.calibration.ac(14) print("done")`

AC Cal Step 15: 1A @ 1kHz

1. Send the following command:
`SEND dmm.range = 1`
2. Source 1A @ 1kHz.
3. Send the following command:
`SEND dmm.calibration.ac(15) print("done")`

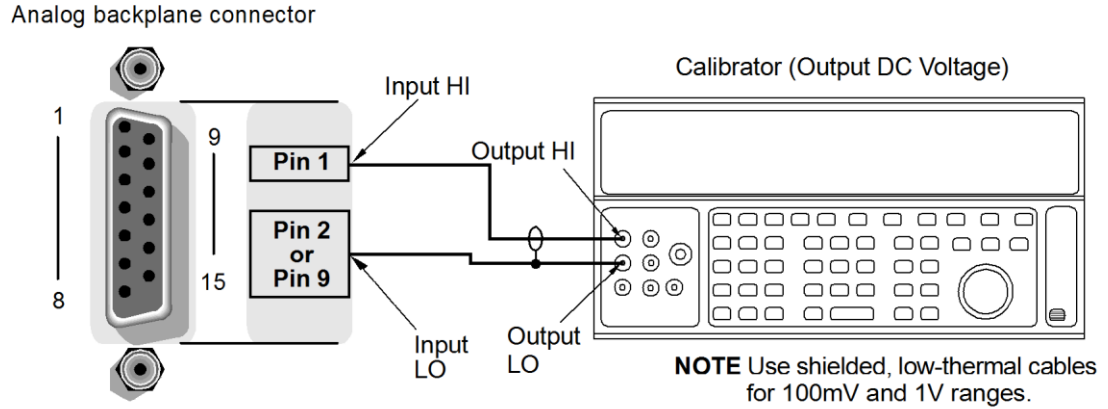
AC Cal Step 16: 2A @ 1kHz

1. Send the following command:
`SEND dmm.range = 2`
2. Source 2A @ 1kHz.
3. Send the following command:
`SEND dmm.calibration.ac(16) print("done")`

Frequency calibration

Make the connections as shown below, then perform the following calibration steps (AC Cal Step 17 and Step 18):

Figure 15-7: Low frequency calibration



AC Cal Step 17: 1V @ 10Hz (factory cal only)

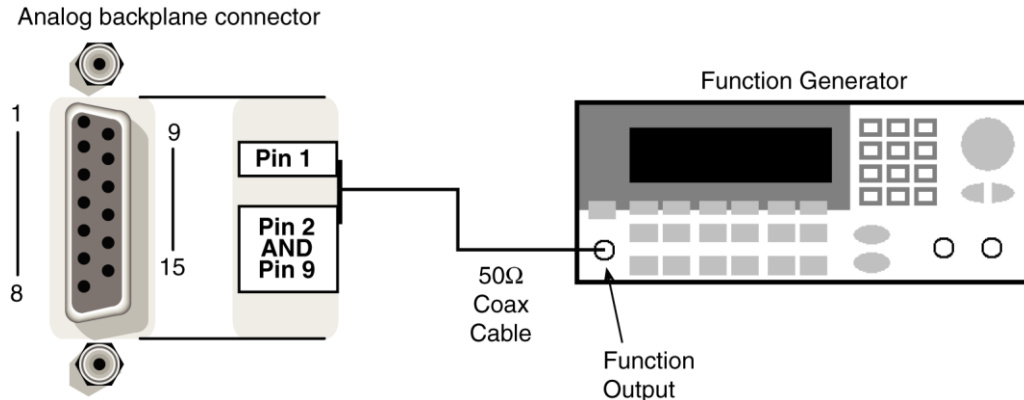
1. Send the following commands:


```
SEND dmm.func = dmm.ac_volts
SEND dmm.range = 1
```
2. Source 1V @ 10Hz.
3. Send the following command:


```
SEND dmm.calibration.ac(17,1) print("done")
```

AC Cal Step 18: 1V @ 1kHz (factory cal only)

Figure 15-8: Frequency verification



1. Source 1V @ 1kHz
2. Send the following command:

```
SEND dmm.calibration.ac(18,1000) print("done")
```

Save calibration

Program today's date, cal due date, and SN, and save the calibration constants in EEPROM (electrically erasable programmable read-only memory) by sending the following commands:

```
dmm.adjustment.date=os.time()  
dmm.calibration.save()  
dmm.calibration.verifydate=dmm.adjustment.date  
dmm.calibration.lock()  
dmm.reset()
```

NOTE Calibration is complete after is has been saved and locked.

Maintenance

In this section:

Introduction	16-1
Fuse replacement	16-1
Front panel tests	16-3

Introduction

The information in this section deals with routine maintenance of Keithley Instruments Series 3700 System Switch Multimeter instruments that can be performed by the operator.

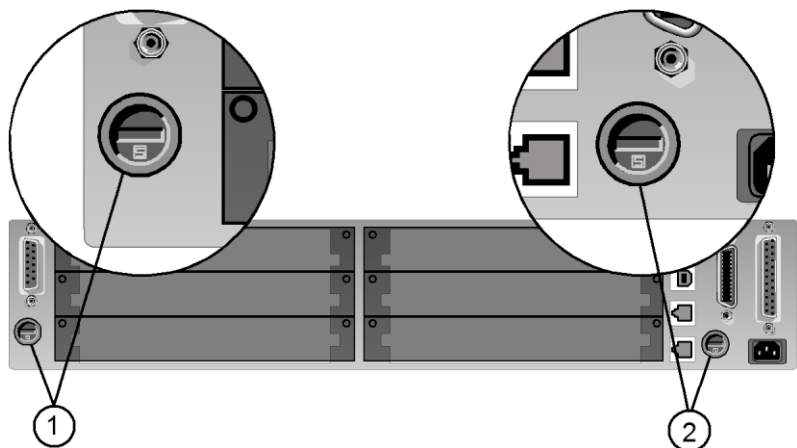
Fuse replacement

The analog backplane AMPS fuse (see item 1 in Fuse location figure) is accessible from the rear panel, just below the analog backplane connector. The instrument fuse (see item 2 in Fuse location figure) is accessible from the rear panel, below the GPIB Connector.

WARNING *Disconnect all external power from the equipment and the line cord before performing any maintenance on the Series 3700.*

Failure to disconnect all power may expose you to hazardous voltages, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

Figure 16-1: Fuse location



Fuse location	Rating	Keithley Instruments part number
(1) Analog backplane fuse	250V, 3A fast blow 5x20mm	FU-99-1
(2) Instrument fuse	250V / 1.25A slow blow 5x20mm	FU-106-1.25

To replace a fuse:

1. Using a flat-tip screwdriver, disengage the fuse holder by rotating it counter-clockwise.
2. Pull out the fuse holder and replace the fuse with the correct type (see table).
3. Reinstall the fuse holder.

If the fuse continues to blow, a circuit malfunction exists and must be corrected. Return the unit to Keithley Instruments for repair.

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.

Test procedure

The front panel keys test lets you check the functionality of each front panel key.

To run the front panel keys test:

1. Display the MAIN MENU by pressing the **MENU** key.
2. Turn the navigation wheel to scroll to the **DISPLAY** menu item and press the **ENTER** key to select.
3. Press the **ENTER** key to select **TEST**.
4. Select **KEYS** or **DISPLAY-PATTERNS** and press the **ENTER** key to run the test.
 - **KEYS:** 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. Press the **EXIT** key twice to end the test.
 - **DISPLAY-PATTERNS:** There are three parts to the display patterns test. Each time **ENTER** or the navigation wheel is pressed, the next part of the test sequence is selected. The three parts of the test sequence are as follows:
 - a. Checkerboard pattern and the annunciators that are on during normal operation.
 - b. Checkerboard pattern (alternate pixels on) and all annunciators.
 - c. Each digit (and adjacent annunciator) is sequenced. All of the pixels of the selected digit are on.
5. Press the **EXIT** key to end the test.
6. Continue pressing the **EXIT** key to back out of the menu structure.

Error and status messages

In this section:

Introduction	17-1
Error summary	17-1
Error effects on scripts	17-1
Reading errors	17-2
Error and status message list.....	17-2

Introduction

This section includes information on error levels, how to read errors, and a complete listing of error messages.

Error summary

Error and status messages are defined in [Error and status message list](#) (on page 17-2). Each message is assigned a level of severity, as 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 will prevent the script/command from being executed.

Reading errors

When errors occur, the error messages will be placed in the error queue. Use error queue commands to request error message information. For example, the following commands request the next complete error information from the error queue and returns the message portion of the error:

```
errorcode, message, severity, node = errorqueue.next()
print(message)
```

The following table lists the commands associated with the error queue.

Error queue command	Description
<code>errorqueue.clear()</code>	Clear error queue of all errors
<code>errorqueue.count</code>	Number of messages in the error queue
<code>errorqueue.next()</code>	Request next error message from queue

Error and status message list

Error number	Error level	Error message
-430	RECOVERABLE	Query deadlocked
-420	RECOVERABLE	Query unterminated
-410	RECOVERABLE	Query interrupted
-363	RECOVERABLE	Input buffer over-run
-360	RECOVERABLE	Communication error
-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

Error number	Error level	Error message
-203	RECOVERABLE	Command protected
-200	RECOVERABLE	Execution error
-154	RECOVERABLE	String too long
-151	RECOVERABLE	Invalid string data
-144	RECOVERABLE	Character data too long
-141	RECOVERABLE	Invalid character data
-140	RECOVERABLE	Character data error
-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
-100	RECOVERABLE	Command error
0	NO_SEVERITY	Queue is empty
603	RECOVERABLE	Power on state lost
605	RECOVERABLE	Calibration dates lost
820	RECOVERABLE	Parsing value
900	FATAL	Internal system
1100	RECOVERABLE	Command unavailable
1101	RECOVERABLE	Parameter too big
1102	RECOVERABLE	Parameter too small
1103	RECOVERABLE	Min greater than max
1104	RECOVERABLE	Too many digits for param type
1107	RECOVERABLE	Cannot modify factory menu
1108	RECOVERABLE	Menu name does not exist
1109	RECOVERABLE	Menu name already exists
1112	RECOVERABLE	Password entered does not match current password
1114	RECOVERABLE	Settings conflict with %s, where %s represents specifics on what the conflict is
1115	RECOVERABLE	Parameter error %s, where %s explains why parameter error

Error number	Error level	Error message
1116	RECOVERABLE	Configuration error %s, where %s explains why configuration error
1200	RECOVERABLE	TSP-Link initialization failed
1201	RECOVERABLE	TSP-Link initialization failed
1202	RECOVERABLE	TSP-Link initialization failed
1203	RECOVERABLE	TSP-Link initialization failed (possible loop in node chain)
1204	RECOVERABLE	TSP-Link initialization failed
1205	RECOVERABLE	TSP-Link initialization failed (no remote nodes found)
1206	RECOVERABLE	TSP-Link initialization failed
1207	RECOVERABLE	TSP-Link initialization failed
1208	RECOVERABLE	TSP-Link initialization failed
1209	RECOVERABLE	TSP-Link initialization failed
1210	RECOVERABLE	TSP-Link initialization failed (node ID conflict)
1211	RECOVERABLE	Node %u is inaccessible, where %u represents a number
1212	RECOVERABLE	Invalid node ID
1213	RECOVERABLE	TSP-Link session expired
1214	RECOVERABLE	TSP-Link unknown remote command encoding
1215	RECOVERABLE	Code execution requested within the local group
1216	RECOVERABLE	Remote execution requested on node in group with pending overlapped operations
1217	RECOVERABLE	Remote execution requested on node outside the local group
1400	RECOVERABLE	Expected at least %d parameters, where %d represents a number
1401	RECOVERABLE	Parameter %d is invalid, where %d represents a number
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
1600	RECOVERABLE	Maximum GPIB message length exceeded
1601	RECOVERABLE	GPIB input queue overrun
1800	RECOVERABLE	Invalid digital trigger mode

Error number	Error level	Error message
1801	RECOVERABLE	Invalid digital I/O line
1802	RECOVERABLE	Digital bit in parameter write protected
2100	FATAL	Could not open socket
2101	FATAL	Could not close socket
2102	RECOVERABLE	LAN configuration already in progress
2103	RECOVERABLE	LAN disabled
2104	RECOVERABLE	Socket error
2105	RECOVERABLE	Unreachable gateway
2106	RECOVERABLE	Could not acquire ip address
2107	RECOVERABLE	Duplicate IP address detected
2108	RECOVERABLE	DHCP lease lost
2109	RECOVERABLE	LAN cable disconnected
2110	RECOVERABLE	Could not resolve hostname
2111	RECOVERABLE	DNS name (FQDN) too long
2112	RECOVERABLE	Connection not established
2200	RECOVERABLE	File write error
2201	RECOVERABLE	File read error
2202	RECOVERABLE	Cannot close file
2203	RECOVERABLE	Cannot open file
2204	RECOVERABLE	Directory not found
2205	RECOVERABLE	File not found
2206	RECOVERABLE	Cannot read current working directory
2207	RECOVERABLE	Cannot change directory
2208	RECOVERABLE	Cannot create directory
2209	RECOVERABLE	Cannot remove directory
2210	RECOVERABLE	File is not a valid script format
2211	RECOVERABLE	File system error
2212	RECOVERABLE	File system command not supported
2213	RECOVERABLE	Too many open files
2214	RECOVERABLE	File access denied
2215	RECOVERABLE	Invalid file handle
2216	RECOVERABLE	Invalid drive
2217	RECOVERABLE	File system busy

Error number	Error level	Error message
2218	RECOVERABLE	Disk full
2219	RECOVERABLE	File corrupt
2220	RECOVERABLE	File already exists
2221	RECOVERABLE	File seek error
2222	RECOVERABLE	End-of-file error
2223	RECOVERABLE	Directory not empty
2300	RECOVERABLE	Upgrade found not upgradable
2301	RECOVERABLE	Upgrade uncompress failed
2302	RECOVERABLE	Upgrade device not ready
2303	RECOVERABLE	Upgrade device type not acceptable
2304	RECOVERABLE	Upgrade write to device checksum failure
2305	RECOVERABLE	Upgrade write to device failed
2306	RECOVERABLE	Upgrade timeout connect with device
2307	RECOVERABLE	Upgrade failure
2400	RECOVERABLE	Invalid specified connection
2401	RECOVERABLE	Invalid timeout seconds (.001 to 30)
2402	RECOVERABLE	TSPnet remote error: %s, where %s explains the remote error
2403	RECOVERABLE	TSPnet failure
2404	RECOVERABLE	TSPnet read failure
2405	RECOVERABLE	TSPnet read failure, aborted
2406	RECOVERABLE	TSPnet read failure, timeout
2407	RECOVERABLE	TSPnet write failure
2408	RECOVERABLE	TSPnet write failure, aborted
2409	RECOVERABLE	TSPnet write failure, timeout
2410	RECOVERABLE	TSPnet max connections reached
2411	RECOVERABLE	TSPnet connection failed
2412	RECOVERABLE	TSPnet invalid termination
2413	RECOVERABLE	TSPnet invalid reading buffer table
2414	RECOVERABLE	TSPnet invalid reading buffer index range
2415	RECOVERABLE	TSPnet feature only supported on TSP connections
2416	RECOVERABLE	TSPnet must specify both port and init
2417	RECOVERABLE	TSPnet disconnected by other side

Error number	Error level	Error message
4900	RECOVERABLE	Reading buffer index %d is invalid, where %d represents a number
4901	RECOVERABLE	The maximum index for this buffer is %d, where %d represents a number
4902	RECOVERABLE	Reading buffers must be able to contain at least one element
4903	RECOVERABLE	Reading buffer expired
4904	RECOVERABLE	ICX parameter count mismatch, %s (Line #%d), where %s and %d provide more information on error
4905	RECOVERABLE	ICX parameter invalid value, %s (Line #%d), where %s and %d provide more information on error
4906	RECOVERABLE	ICX invalid function id, %s (Line #%d), where %s and %d provide more information on error
4907	RECOVERABLE	Cannot modify built-in reading buffers
4908	RECOVERABLE	Cannot change this setting unless buffer is cleared
4909	RECOVERABLE	Reading buffer not found within device
4910	RECOVERABLE	No readings exist within buffer
4911	RECOVERABLE	Table not found within buffer
4912	RECOVERABLE	Attribute not found within buffer
4914	RECOVERABLE	Index exceeds maximum readings stored in buffer
4915	RECOVERABLE	Attempting to store past capacity of reading buffer
5500	RECOVERABLE	Card unknown error
5501	RECOVERABLE	Failed card NVMEM write
5502	RECOVERABLE	Failed card NVMEM read
5503	RECOVERABLE	Closure count lost
5504	RECOVERABLE	Temperature sensor failure
5505	RECOVERABLE	Error completing a card action in requested operation
5506	RECOVERABLE	Communication error with a card in requested operation
5507	RECOVERABLE	Card operation completed under low total power
5508	RECOVERABLE	Card operation completed under low bank power
5509	RECOVERABLE	Card operation completed under low slot power
5510	RECOVERABLE	Not enough total power to hold requested card operation
5511	RECOVERABLE	Not enough bank power to hold requested card operation
5512	RECOVERABLE	Not enough slot power to hold requested card operation

Error number	Error level	Error message
5513	RECOVERABLE	Not enough total power to complete requested card operation
5514	RECOVERABLE	Not enough bank power to complete requested card operation
5515	RECOVERABLE	Not enough slot power to complete requested card operation
5516	RECOVERABLE	Slot empty, no configuration data exist
5517	RECOVERABLE	Slot error, configuration data not found
5518	RECOVERABLE	Slot error, communication error accessing configuration data
5519	RECOVERABLE	Slot error, timeout error accessing configuration data
5520	RECOVERABLE	Channel error, channel list contains a channel not in system
5521	RECOVERABLE	Parameters adjusted, must recreate scan
5522	RECOVERABLE	Scan running, must abort scan
5600	RECOVERABLE	10 vdc zero error
5601	RECOVERABLE	100 vdc zero error
5602	RECOVERABLE	10 vdc full scale error
5603	RECOVERABLE	-10 vdc full scale error
5604	RECOVERABLE	100 vdc full scale error
5605	RECOVERABLE	100m vdc zero error
5606	RECOVERABLE	100 2-w zero error
5607	RECOVERABLE	10k 2-w zero error
5608	RECOVERABLE	100k 2-w zero error
5609	RECOVERABLE	10M 2-w zero error
5610	RECOVERABLE	10M 2-w full scale error
5611	RECOVERABLE	10M 2-w open error
5612	RECOVERABLE	100 4-w zero error
5613	RECOVERABLE	10k 4-w zero error
5614	RECOVERABLE	100k 4-w zero error
5615	RECOVERABLE	10M 4-w sense lo zero error
5616	RECOVERABLE	1k 4-w full scale error
5617	RECOVERABLE	10k 4-w full scale error
5618	RECOVERABLE	100k 4-w full scale error
5619	RECOVERABLE	1M 4-w full scale error

Error number	Error level	Error message
5620	RECOVERABLE	10M 4-w full scale error
5621	RECOVERABLE	10m adc zero error
5622	RECOVERABLE	100m adc zero error
5623	RECOVERABLE	10m adc full scale error
5624	RECOVERABLE	100m adc full scale error
5625	RECOVERABLE	1 adc full scale error
5626	RECOVERABLE	2k 4-w dckt loff zero error
5627	RECOVERABLE	2k 4-w dckt lon zero error
5628	RECOVERABLE	1k 4-w dckt loff zero error
5629	RECOVERABLE	1k 4-w dckt lon zero error
5630	RECOVERABLE	100 4-w dckt loff zero error
5631	RECOVERABLE	100 4-w dckt lon zero error
5632	RECOVERABLE	10 4-w dckt loff zero error
5633	RECOVERABLE	10 4-w dckt lon zero error
5634	RECOVERABLE	1 4-w dckt lon zero error
5635	RECOVERABLE	10 2-w zero error
5636	RECOVERABLE	10 4-w full scale error
5637	RECOVERABLE	100 4-w full scale error
5638	RECOVERABLE	10u adc zero error
5639	RECOVERABLE	100u adc zero error
5640	RECOVERABLE	1m adc zero error
5641	RECOVERABLE	1 adc zero error
5642	RECOVERABLE	10u adc full scale error
5643	RECOVERABLE	100u adc full scale error
5644	RECOVERABLE	1m adc full scale error
5645	RECOVERABLE	1 vac fast noise error
5646	RECOVERABLE	1 vac fast full scale error
5647	RECOVERABLE	100m vac dac error
5648	RECOVERABLE	1 vac dac error
5649	RECOVERABLE	10 vac dac error
5650	RECOVERABLE	100 vac dac error
5651	RECOVERABLE	100m vac zero error
5652	RECOVERABLE	100m vac full scale error

Error number	Error level	Error message
5653	RECOVERABLE	1 vac zero error
5654	RECOVERABLE	1 vac full scale error
5655	RECOVERABLE	1 vac noise error
5656	RECOVERABLE	10 vac zero error
5657	RECOVERABLE	10 vac full scale error
5658	RECOVERABLE	10 vac noise error
5659	RECOVERABLE	100 vac zero error
5660	RECOVERABLE	100 vac full scale error
5661	RECOVERABLE	300 vac zero error
5662	RECOVERABLE	300 vac full scale error
5663	RECOVERABLE	300 vac noise error
5664	RECOVERABLE	Post filter offset error
5665	RECOVERABLE	1 aac zero error
5666	RECOVERABLE	1 aac full scale error
5667	RECOVERABLE	3 aac zero error
5668	RECOVERABLE	3 aac full scale error
5669	RECOVERABLE	1V 10 Hz amplitude error
5670	RECOVERABLE	Frequency gain error
5671	RECOVERABLE	100 Ohm loff Ocomp FS error
5672	RECOVERABLE	10k Ohm loff Ocomp FS error
5673	RECOVERABLE	Temperature cold cal error
5674	RECOVERABLE	Analog output zero error
5675	RECOVERABLE	Analog output pos. gain error
5676	RECOVERABLE	Analog output neg. gain error
5677	RECOVERABLE	100 4-w dckt loff full scale error
5678	RECOVERABLE	100 4-w dckt lon full scale error
5679	RECOVERABLE	10 4-w dckt full scale error
5680	RECOVERABLE	1 4-w dckt lon full scale error
5681	RECOVERABLE	10k 4-w ocomp loff full scale error
5682	RECOVERABLE	10k 4-w ocomp lon full scale error
5683	RECOVERABLE	2k 4-w dckt loff full scale error
5684	RECOVERABLE	2k 4-w dckt lon full scale error
5685	RECOVERABLE	1k 4-w dckt loff full scale error

Error number	Error level	Error message
5686	RECOVERABLE	1k 4-w dckt lon full scale error
5687	RECOVERABLE	10 4-w zero error
5688	RECOVERABLE	10 4-w loff zero error
5689	RECOVERABLE	1m aac full scale error
5690	RECOVERABLE	1m aac zero error
5691	RECOVERABLE	10m aac full scale error
5692	RECOVERABLE	10m aac zero error
5693	RECOVERABLE	100m aac full scale error
5694	RECOVERABLE	100m aac zero error
5695	RECOVERABLE	Offset calibration error
5696	RECOVERABLE	1V 10 Hz frequency error
5697	RECOVERABLE	Calibration data invalid
5698	RECOVERABLE	AC calibration data lost
5699	RECOVERABLE	DC calibration data lost
5700	RECOVERABLE	PreCal calibration data lost
5701	RECOVERABLE	A/D timeout
5702	RECOVERABLE	1 4-w dckt loff zero error
5703	RECOVERABLE	100 4-w loff zero error
5704	RECOVERABLE	10k 4-w loff zero error
5705	RECOVERABLE	10 4-w dckt loff full scale error
5706	RECOVERABLE	1 4-w dckt loff full scale error
5707	RECOVERABLE	1k TRTD HI lon zero error
5708	RECOVERABLE	1k TRTD HI loff zero error
5709	RECOVERABLE	1k TRTD SLO lon zero error
5710	RECOVERABLE	1k TRTD SLO loff zero error
5711	RECOVERABLE	10k TRTD HI lon zero error
5712	RECOVERABLE	10k TRTD HI loff zero error
5713	RECOVERABLE	10k TRTD SLO lon zero error
5714	RECOVERABLE	10k TRTD SLO loff zero error
5715	RECOVERABLE	100k TRTD HI lon zero error
5716	RECOVERABLE	100k TRTD SLO lon zero error
5717	RECOVERABLE	1k TRTD HI lon full scale error
5718	RECOVERABLE	1k TRTD HI loff full scale error

Error number	Error level	Error message
5719	RECOVERABLE	1k TRTD SLO Ion full scale error
5720	RECOVERABLE	1k TRTD SLO Ioff full scale error
5721	RECOVERABLE	10k TRTD HI Ion full scale error
5722	RECOVERABLE	10k TRTD HI Ioff full scale error
5723	RECOVERABLE	10k TRTD SLO Ion full scale error
5724	RECOVERABLE	10k TRTD SLO Ioff full scale error
5725	RECOVERABLE	100k TRTD HI Ion full scale error
5726	RECOVERABLE	100k TRTD SLO Ion full scale error
5727	RECOVERABLE	10 vdc full scale 6p4 error
5728	RECOVERABLE	10 vdc full scale p64 error
5729	RECOVERABLE	10 vdc zero 6p4 error
5730	RECOVERABLE	10 vdc zero p64 error
5731	RECOVERABLE	1k 4-w ocomp Ioff full scale error
5732	RECOVERABLE	Questionable calibration
5733	RECOVERABLE	Questionable temperature
5734	RECOVERABLE	Internal DMM system error
5735	RECOVERABLE	General unknown DMM error
5736	RECOVERABLE	Untranslated DMM error
5737	RECOVERABLE	Error completing DMM action in requested operation
5738	RECOVERABLE	Communication error with DMM in requested operation
5739	RECOVERABLE	DMM calibration error occurred during processing command
5740	RECOVERABLE	DMM calibration error occurred setting adjustment date
5741	RECOVERABLE	DMM calibration error occurred getting adjustment date
5742	RECOVERABLE	DMM calibration error occurred setting verify date
5743	RECOVERABLE	DMM calibration error occurred getting verify date
5744	RECOVERABLE	DMM calibration error occurred setting password
5745	RECOVERABLE	DMM calibration error occurred getting password
5746	RECOVERABLE	DMM calibration error occurred setting count
5747	RECOVERABLE	DMM calibration error occurred getting count

IEEE-1588 Glossary of Terms

In this appendix:

Boundary clock.....	A-1
Epoch	A-1
Grandmaster clock	A-1
Master clock	A-2
PTP	A-2
PTP port	A-2
PTP subdomain.....	A-2

Boundary clock

A clock with more than a single PTP port, with each PTP port providing access to a separate PTP communication path. Boundary clocks are used to eliminate timing fluctuations caused by routers and other network elements.

Definition derived from NIST [website](http://ieee1588.nist.gov) (<http://ieee1588.nist.gov>).

Epoch

The reference time defining the origin of a time scale. For PTP, the epoch is January 1, 1970.

Definition derived from NIST [website](http://ieee1588.nist.gov) (<http://ieee1588.nist.gov>).

Grandmaster clock

Serves as the primary reference time to which all other clocks are ultimately synchronized. If there is synchronization across multiple subnetworks, the grandmaster clock is the reference of time for all subnetworks.

Definition derived from NIST [website](http://ieee1588.nist.gov) (<http://ieee1588.nist.gov>).

Master clock

Within a region (on the same subnetwork), the master clock is the clock that serves as a primary source of time.

Definition derived from NIST [website](http://ieee1588.nist.gov) (<http://ieee1588.nist.gov>).

PTP

Precision Time Protocol, synonymous with IEEE-1588.

Definition derived from NIST [website](http://ieee1588.nist.gov) (<http://ieee1588.nist.gov>).

PTP port

A PTP port is the logical access point for IEEE-1588 communications to the clock containing the port.

Definition derived from NIST [website](http://ieee1588.nist.gov) (<http://ieee1588.nist.gov>).

PTP subdomain

A logical grouping of 1588 clocks that synchronize to each other using PTP protocol but are not necessarily synchronized to PTP clocks in another subdomain. Allows a single common network with independent groups of synchronized devices.

Definition derived from NIST [website](http://ieee1588.nist.gov) (<http://ieee1588.nist.gov>).

Index

1

1-OHM and 10-OHM resistance ranges, verifying • 14-24

2

2-wire

resistance verification data • 14-21

verifying • 14-21

4

4-wire

resistance verification data • 14-20

short applied verification data • 14-26

short, verifying zeros • 14-25

verifying • 14-19

A

AC current

1mA to 1A ranges, verification data • 14-16

1mA to 3A ranges, verifying • 14-15

3A range, verification data • 14-17

calibration • 15-13

AC voltage

verifying • 14-9

AC volts calibration • 15-11

acceptor trigger mode • 8-22, 8-27

access recall attributes example • 7-16

action keys

action keys • 4-34

CLOSE • 4-33

OPEN • 4-34

OPEN ALL • 4-33

RATE • 4-34

RECall • 4-34

STORE • 4-35

ACV

verification data • 14-10

alarms

scheduling • 11-5, 11-8

anonymous script • 2-25

APERTURE • 4-21

appending readings • 7-7

Arm Action trigger • 8-6

arrays, TSL • 2-36

- assigning a value to an attribute •
13-3
- assigning groups • 3-5
- attributes • 13-2, 13-3
 - reading • 13-4
- AUTO key • 4-32
- auto ranging • 5-4
 - AUTODELAY • 4-22
 - over front panel • 5-3
- Autoexec script • 2-26
- AUTORANGE • 4-22
- autorun scripts • 2-26
- AUTOZERO • 4-22

B

- background scan execution • 8-3
- bandwidth • 5-7
- base library functions • 2-43
- basic front panel REL procedure •
6-2
- basic reciprocal operation • 6-8
- basic scan procedure • 8-7
- beeper functions and attributes •
13-11, 13-16
- bit functions • 13-11, 13-17
- bit operations • 13-17
- boundary clock • A-1
- branching • 2-40

- buffer • 8-9
 - configuration (front panel) • 7-6
 - data store commands • 7-8
 - for...do loops • 7-19
 - overview • 7-1
 - programming examples • 7-13
 - reading attributes • 7-14
 - reading buffer • 7-12
 - read-only attributes • 7-13
 - recall attributes • 7-14
 - remote operation • 7-7
 - status • 7-16
 - storage control attributes • 7-12

- bus operation

- scanning • 8-12

C

- calculating
 - reading limits • 14-4
- calculations
 - math • 6-3
- calibration • 15-1, 15-4
 - considerations • 15-3
 - cycle • 15-3
 - DC current • 15-9
 - saving • 15-16
- CHAN key • 4-17

- configuration • 4-11
 - channel
 - assignments • 8-2
 - display • 4-4, 7-6
 - existing scan • 8-9
 - functions and attributes • 13-11, 13-24
 - range • 5-3
 - scanning limitations • 4-1
 - type indications • 4-8
 - Channel Action Trigger • 8-6
 - characters, wild • 13-1
 - chunk • 2-6
 - chunks
 - multiple • 2-6
 - multiple-statement • 2-6
 - non-scripted, executing • 2-15
 - single statement • 2-6
 - clearing
 - readings • 7-4
 - registers and queues • 12-9
 - clock
 - grandmaster • A-1
 - master • A-2
 - CLOSE key • 4-33
 - command
 - programming notes • 13-1
 - queries • 2-3
 - table entries • 9-9
 - TSP advanced features • 3-10
 - concatenation • 2-39
 - CONFIG key • 4-11
 - configuration
 - configuration • 4-11, 6-10, 13-8
 - configuration (front panel) • 7-6
 - connection
 - line power • 14-3
 - considerations
 - calibration • 15-3
 - test • 14-6
 - contact information • 1-1
 - counts • 8-7
 - current verification data
 - 10 μ A to 100 μ A ranges • 14-12
 - 1mA to 3A ranges • 14-14
 - CURSOR keys • 4-32
 - cycle, calibration • 15-3
- D**
- data storage and retrieval, buffer • 7-1
 - data store (buffer) commands • 7-8
 - dataqueue functions and attributes • 13-11, 13-85

- date values • 7-16
- dB
 - channel type indication • 4-8
 - configuration • 6-10
 - DBREF • 4-22
 - non-switch channels • 4-9
 - scanning • 6-11
- DBREF • 4-22
- DC
 - voltage verification data • 14-7
 - volts calibration • 15-6
- DC current
 - 10 μ A to 100 μ A ranges, verification data • 14-12
 - 10 μ A to 100 μ A ranges, verifying • 14-11
 - 1mA to 3A ranges, verification data • 14-14
 - 1mA to 3A ranges, verifying • 14-13
- DC voltage • 14-6, 14-7, 15-6
- default file extensions • 9-1
- delay functions • 13-11, 13-86
- DELETE key • 4-20
- DETECTBW • 4-22
- digio functions and attributes • 13-11, 13-87
- digital
 - filter types • 5-8
 - filter window • 5-10
 - I/O channel indication • 4-8
- DIGITS • 4-22
- digits ICL programming • 5-4
- digits, setting • 5-4
- discrete resistance verification data • 14-24
- display • 4-4, 7-6
 - unit serial number • 1-3
 - user-defined messages • 2-16
- display functions and attributes • 13-11, 13-93
- DISPLAY key • 4-16
- DMM
 - attributes, existing scan • 8-9
 - configuration • 13-8
 - key • 4-21
 - key configuration • 4-21
 - new configuration • 13-8
- dmm functions and attributes • 13-11, 13-109
- dry circuit resistance • 14-6
 - verification data • 14-23
 - verifying • 14-22
- DRYCIRCUIT • 4-22
- dynamic buffer programming example • 7-18

- dynamically-allocated buffers • 7-17
- E**
- either edge trigger mode • 8-23
 - ENTER key • 4-25
 - environmental conditions • 14-2
 - epoch • A-1
 - equipment
 - recommended • 15-3
 - recommended test • 14-3
 - error effects, scripts • 17-1
 - errorqueue functions and attribute • 13-12, 13-176
 - errors
 - and status message list • 17-1, 17-2
 - effects on scripts • 17-1
 - queue • 12-26
 - summary • 17-1
 - event blenders • 8-3
 - event register • 12-19
 - event registers, system summary • 12-16
 - event status register • 12-5
 - eventlog functions and attributes • 13-12, 13-177
 - events • 8-2
 - example applications
 - in Series 3700-based systems • 11-7
 - examples
 - access recall attributes example • 7-16
 - dynamic buffer programming example • 7-18
 - exceeding reading buffer capacity • 7-21
 - new configuration • 13-8
 - passing parameter • 13-4
 - reading limit calculation • 14-4
 - scanning • 8-14
 - script • 10-3
 - variable assignment • 13-4
 - exceeding reading buffer capacity example • 7-21
 - exit functions • 13-12, 13-180
 - EXIT key • 4-25
- F**
- factory defaults, restoring • 14-5
 - falling edge trigger mode • 8-20
 - file • 9-1
 - formats • 9-1
 - functions • 13-12, 13-181
 - I/O • 9-3
 - system navigation • 9-2
 - FILTER
 - key • 4-25

- key configuration • 4-22, 4-25
- filter, digital • 5-8
 - characteristics • 5-8
 - overview • 5-8
 - repeating average • 5-9
- foreground scan execution • 8-3
- format attributes • 13-12, 13-183
- frequency
 - calibration • 15-15
 - verification data • 14-18
 - verifying • 14-18
- front panel
 - introduction • 4-1, 4-4
 - operation • 7-2
 - scanning • 8-10
 - tests • 16-3
- fs functions • 13-12, 13-186
- FUNC
 - key • 4-26
 - key configuration • 4-26
 - menu • 4-22
- FUNC key • 4-26
- functions • 2-35, 13-2
- fuse replacement • 16-1

G

- general bus command sequence • 12-14

GPIB

- address • 2-12
- attributes • 13-12, 13-187
- interface connection • 2-11

- grandmaster clock • A-1

groups

- assigning • 3-5
- coordinating remote • 3-7
- different test scripts • 3-4
- leader • 3-5
- reassigning • 3-6

I

ICL

- general device control • 10-5
- ICL commands • 8-12, 13-11
- TSP-specific device control • 10-12

- idle state • 8-5

IEEE-1588

- configuring • 11-3
- enabling • 11-3
- implementation in Series 3700 • 11-1
- in Series 3700-based systems • 11-7

introduction • 11-1

INPUTDIV • 4-23

INSERT key • 4-26

instruments, synchronizing multiple
• 11-9

interactive script • 2-19

interface connection • 2-12

K

key configuration • 4-21

keys

- action keys • 4-34
- AUTO • 4-32
- CHAN • See CHAN key
- CLOSE • 4-33
- CONFIG • 4-11
- CURSOR • 4-32
- DELETE • 4-20
- DISPLAY • 4-16
- ENTER • 4-25
- EXIT • 4-25
- FILTER • 4-25
- FUNC • 4-26
- INSERT • 4-26
- key • 4-21
- LIMIT • 4-27
- LOAD • 4-27

MENU • 4-28

OPEN • 4-34

OPEN ALL • 4-33

PATT • 4-29

RANGE • 4-33

RATE • 4-34

RECall • 4-34

REL • 4-30

RUN • 4-30

SCAN • 4-31

SLOT • 4-32

STORE • 4-35

TRIG • 4-32

L

LAN • 2-12

libraries, standard • 2-42

LIMIT

- key • 4-27
- key configuration • 4-27
- menu • 4-23

limit number (Y) • 13-1

line power • 14-3

LINESYNC • 4-23

LOAD key • 4-27

local state • 2-32

- differences • 2-32

localnode functions and attributes •
13-13, 13-210

logical

instruments • 13-5

operators • 2-38

logical operations • 2-38

loop control • 2-41

LXI

event log • 11-7

LXI Class B triggering (IEEE-
1588) • 11-1

M

maintenance • 16-1

makegetter functions • 13-13

manual range keys • 5-2

master clock • A-2

master node overview • 3-5

master trigger mode, rising edge •
8-21, 8-25

math

calculations • 6-3

library functions • 2-45

MATH • 4-23

matrix card notation • 13-24

measure

and switching capabilities • 1-2

capabilities • 1-2

voltage • 2-15

Measure Action Trigger • 8-6

measurement

event registers • 12-24

maximum readings • 5-1

ranges • 5-1

Measurement event register
(measurement) • 12-8

memory

functions • 13-13, 13-219

nonvolatile • 2-8

MENU key • 4-28

modules

channel assignments • 8-2

monitoring alarms • 11-6

moving average filter • 5-9

multiple

chunks • 2-6

statement chunk • 2-6

mX+b • 6-4

mX+b REL • 6-4

N

named scripts

overview • 2-5

RUN • 2-25

saving • 2-23

- navigation wheel • 4-33
- negative transition registers • 12-2
- new configuration example, DMM • 13-8
- node
 - master overview • 3-5
 - TSP-Link • 13-5
- node enable registers, controlling • 12-16
- non-scripted chunks, sending • 2-15
- nonvolatile memory • 2-8
- NPLC • 4-23
- O**
- OFFSETCOMP • 4-23
- opc functions • 13-13, 13-220
- OPEN ALL key • 4-33
- OPEN key • 4-34
- OPENDETECT • 4-24
- operation event registers • 12-6, 12-21
- operation keys
 - CHAN • See CHAN key
 - DELETE • 4-20
 - ENTER • 4-25
 - EXIT • 4-25
 - FILTER • 4-25
 - FUNC • 4-26
 - INSERT • 4-26
 - key • 4-21
 - LIMIT • 4-27
 - LOAD • 4-27
 - MENU • 4-28
 - PATT • 4-29
 - RECall • 4-34
 - REL • 4-30
 - RUN • 4-30
 - SCAN • 4-31
 - SLOT • 4-32
 - TRIG • 4-32
- operations
 - bit • 13-17
 - logic • 13-17
 - mX+b • 6-4
 - mX+b REL • 6-4
 - reciprocal (1/X) • 6-7
- operators • 2-34
- output queue • 12-2, 12-25
- overlapped operations in remote groups, coordinating • 3-7
- P**
- parallel test scripts • 3-6
- PATT
 - configuration • 4-30

- key • 4-29
- percent • 6-6
- positive transition registers • 12-2
- POWER switch • 4-17
- Precedence • 2-37
- primary node • 2-33
- print functions • 13-13, 13-221
- programming
 - enable registers • 12-10
 - interaction • 2-15
 - script model • 2-9
 - transition registers • 12-10
- Project Navigator • 2-14
- PTP • A-2
 - definition • A-2
 - port • A-2
 - subdomain • A-2
- PTP to UTC, correlating • 11-2
- Q**
- queries • 2-3
- queues • 12-2
- R**
- range
 - manual keys • 5-2
 - remote programming • 5-3
 - selecting auto • 5-4
 - selecting manual • 5-3
- RANGE • 4-24
- RANGE keys • 4-32, 4-33
- RATE key • 4-34, 5-5
- reading buffer • 7-12
 - capacity, exceeding • 7-21
 - creating • 7-2
 - deleting • 7-5
 - described • 7-12
 - designations • 7-12
 - removing stale values • 3-9
 - selecting • 7-3
- reading limit calculation example • 14-4
- readings
 - errors • 17-2
 - RECall • 7-5
 - registers • 12-11
 - saving • 7-3
 - storing • 7-3
- rear panel
 - summary • 2-10
- RECall key • 4-34
- reciprocal (1/X) • 6-7
- registers
 - enable and transition • 12-15

- negative transition • 12-2
- operation events • 12-6, 12-21
- positive transition • 12-2
- questionable events • 12-7, 12-23
- registers, system summary • 12-4
- REL
 - buffer operation • 7-7
 - function • 6-1
 - key • 4-30
 - key configuration • 4-30
 - menu • 4-24
 - remote operation • 6-2
- remote
 - calibration procedure • 15-5
- remote buffer operation • 7-7
- remote state differences • 2-32
- repeating average filter • 5-9
- requirements, verification tests • 14-2
- reset functions • 13-13, 13-230
- RESET switch • 4-17
- resistance
 - calibration • 15-8
 - reading limits • 14-4
 - verifying • 14-19, 14-21, 14-22

- resistance ranges (1-OHM and 10-OHM), verifying • 14-24
- rising edge
 - acceptor trigger mode • 8-22
 - master trigger mode • 8-21
 - trigger mode • 8-21, 8-22

RJ-45

- Ethernet interface connection • 2-12

RUN key • 4-30

run-time environment

- overview • 2-3, 2-8
- script, restoring • 2-31

S

SCAN

- configuration • 4-31
- key • 4-31

scanning

- configuration • 8-11
- counts • 8-7
- digits setting • 5-4
- DMM configuration • 5-3
- examples • 8-14
- execution, foreground and background • 8-3
- functions and attributes • 13-13, 13-230
- fundamentals • 8-1

- math setup • 6-9
- REL value • 6-3
- schedule functions and attributes • 13-14, 13-250
- Script Editor • 2-14
- scripts
 - autoexec • 2-26
 - automatically run • 2-25
 - autorun scripts • 2-26
 - commands, using • 2-17
 - definition • 2-7
 - deleting • 2-31
 - error effects • 17-1
 - examples • 2-17, 9-4, 10-3
 - function, using • 2-18
 - interactive • 2-9, 2-19
 - load only • 2-20
 - loading from front panel • 2-28
 - loading user • 2-24
 - management • 2-30
 - name attribute • 2-22
 - named • 2-5, 2-23, 2-25
 - parallel test, running • 3-6
 - restoring in run-time environment • 2-31
 - retrieving • 2-30
 - running • 2-25, 2-27, 3-6
 - saving • 2-22, 2-23, 2-29
 - Script Editor • 2-14
 - statements, using • 2-17
 - test scripts across the TSP-Link network • 3-8
 - unnamed • 2-25
 - user • 2-16, 2-20, 2-21, 2-22, 2-24, 2-25, 2-27, 2-29, 2-30
- Sequence Action Trigger • 8-6
- serial number • 1-3
- serial polling • 12-14
- service request
 - checking • 12-14
 - register • 12-14
- Service Request Enable Register • 12-14
- setup functions and attribute • 13-14, 13-252
- single statement chunk • 2-6
- slot indicator (X) • 13-1
- SLOT key • 4-32
- slot[X] attributes • 13-14, 13-255
- SPD general bus command sequence • 12-14
- SPE general bus command sequence • 12-14
- SRQ enable registers, controlling • 12-16
- standard

- Ethernet interface connection • 2-12
 - event register • 12-19
 - event status register • 12-5
 - libraries • 2-42
 - standard libraries • 2-42
 - state • 3-9
 - local • 2-32
 - remote • 2-32
 - status byte
 - and system summary • 12-3
 - register • 12-13
 - service request (SRQ) • 12-2, 12-12
 - service request commands • 12-15
 - status function and attributes • 13-14, 13-264
 - status function summary • 12-8
 - status model • 12-1
 - step counts • 8-7
 - STEP key • 4-34
 - STORE
 - key • 4-35
 - key configuration • 4-35
 - string library functions • 2-44
 - summary, test • 14-5
 - switching capabilities • 1-2
 - synchronization
 - multiple instruments • 11-9
 - synchronous
 - acceptor trigger mode • 8-27
 - master trigger mode • 8-25
 - trigger mode • 8-28
 - triggering modes, understanding • 8-24
 - system
 - connections • 2-10
 - system summary
 - and status byte • 12-3
 - event registers • 12-16
 - registers • 12-4
- ## T
- tables • 2-36
 - temperature
 - range • 5-2
 - test
 - considerations • 14-6
 - procedure • 16-3
 - summary • 14-5
 - verification requirements • 14-2
 - Test Script Builder • 2-13
 - Test Script Language Reference • 2-33
 - Test Script Processor • 2-2

- test scripts across the TSP-Link network • 3-8
 - THERMO • 4-24
 - THRESHOLD • 4-24
 - time
 - stamp • 7-5
 - values • 7-16
 - timer functions • 13-14, 13-286
 - totalizer
 - channel type indication • 4-8
 - TRIG key • 4-32
 - trigger functions and attributes • 13-14, 13-287
 - trigger mode
 - either edge • 8-23
 - falling edge • 8-20
 - rising edge acceptor • 8-22
 - rising edge master • 8-21
 - synchronous • 8-28
 - synchronous acceptor • 8-27
 - synchronous master • 8-25
 - syntax rules • 13-4
 - trigger model
 - components • 8-5
 - described • 8-4
 - triggers • 8-6
 - TSL reference • See Test Script Language Reference
 - TSP
 - advanced features • 3-1
 - installing software • 2-10
 - programming levels • 2-8
 - TSP-Link
 - communicating between TSP-enabled instruments • 10-4
 - nodes • 13-5
 - system • 2-32
 - tsplink function and attributes • 13-14, 13-294
 - tsplink.trigger functions and attributes • 13-295
 - tspnet functions and attributes • 13-15, 13-300
- ## U
- UNITS • 4-25
 - unnamed scripts • 2-25
 - upgrade
 - functions • 13-15, 13-309
 - USB
 - connection • 2-13
 - user scripts
 - creating • 2-20
 - creating alternative • 2-21
 - loading • 2-24

- modifying • 2-29
- nonvolatile memory • 2-16
- retrieving • 2-30
- running • 2-25
- running from front panel • 2-27
- writing and loading • 2-16

userstring functions • 3-4, 13-15,
13-310

V

- variables • 2-34
- verification
 - instrument address • 14-2
 - limits • 14-4
 - test procedures • 14-5, 14-6
 - test requirements • 14-2

W

- waitcomplete functions • 13-15, 13-312
- warm-up • 14-2
- wheel, navigation • 4-33
- wild characters • 13-1

Model No. _____ Serial No. _____ Date _____

Name and Telephone No. _____

Company _____

List all control settings, describe problem and check boxes that apply to problem. _____

- | | | |
|--------------------------------------------------|----------------------------------------------------------|--------------------------------------------------------------------------|
| <input type="checkbox"/> Intermittent | <input type="checkbox"/> Analog output follows display | <input type="checkbox"/> Particular range or function bad; specify _____ |
| <input type="checkbox"/> IEEE failure | <input type="checkbox"/> Obvious problem on power-up | <input type="checkbox"/> Batteries and fuses are OK |
| <input type="checkbox"/> Front panel operational | <input type="checkbox"/> All ranges or functions are bad | <input type="checkbox"/> Checked all cables |

Display or output (check one)

- | | |
|-------------------------------------------|--------------------------------------------------------------|
| <input type="checkbox"/> Drifts | <input type="checkbox"/> Unable to zero |
| <input type="checkbox"/> Unstable | <input type="checkbox"/> Will not read applied input |
| <input type="checkbox"/> Overload | |
| <input type="checkbox"/> Calibration only | <input type="checkbox"/> Certificate of calibration required |
| <input type="checkbox"/> 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.



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

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY • www.keithley.com