

Model DAQ6510 Data Acquisition and Multimeter System

Reference Manual

DAQ6510-901-01 Rev. C / June 2024



Mess- und Prüftechnik. Die Experten.

**Ihr Ansprechpartner /
Your Partner:**

dataTec AG

E-Mail: info@datatec.eu

datatec.eu



DAQ6510-901-01C

KEITHLEY

A Tektronix Company

DAQ6510

Data Acquisition and Multimeter System

Reference Manual

© 2024, Keithley Instruments, LLC

Cleveland, Ohio, U.S.A.

All rights reserved.

Any unauthorized reproduction, photocopy, or use of the information herein, in whole or in part, without the prior written approval of Keithley Instruments, LLC, is strictly prohibited.

These are the original instructions in English.

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

The Lua 5.0 software and associated documentation files are copyright © 2003 - 2004, Lua.org, PUC-Rio. You can access terms of license for the Lua software and associated documentation at the Lua licensing site (<https://www.lua.org/license.html>).

Microsoft, Visual C++, Excel, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Document number: DAQ6510-901-01 Rev. C June 2024

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

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

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

The types of product users are:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The  symbol indicates a connection terminal to the equipment frame.

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

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

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

Safety precaution revision as of June 2018.

Table of contents

Introduction.....	1-1
Welcome	1-1
Extended warranty	1-2
Contact information	1-2
Organization of manual sections.....	1-3
Capabilities and features.....	1-4
General ratings.....	1-4
 Installation.....	 2-1
Dimensions	2-1
Bumpers and extension feet	2-3
Removing the bumpers	2-3
Adjusting the tilt feet.....	2-4
Removing the tilt feet for rack mounting	2-4
Installing a switching module	2-4
Instrument power	2-5
Connect the power cord	2-6
Turn the DAQ6510 on or off.....	2-6
Remote communications interfaces	2-7
Supported remote interfaces	2-8
Comparison of the communications interfaces.....	2-8
GPIB setup.....	2-10
USB communications.....	2-16
LAN communications	2-22
RS-232.....	2-32
DAQ6510 web interface	2-33
How to install the Keithley I/O Layer	2-40
Modifying, repairing, or removing Keithley I/O Layer software	2-40
Determining the command set you will use	2-41
Interface access	2-42
Changing the interface access type	2-43
Changing the password	2-44
Switching control interfaces.....	2-45
System information	2-45
View system information from the front panel	2-45
View system information using SCPI commands	2-46
View system information using TSP commands	2-47
 Instrument description	 3-1
Front-panel overview.....	3-1
Rear-panel overview	3-3
Touchscreen display	3-5
Scroll bars	3-5
Enter information.....	3-6

Adjust the backlight brightness and dimmer	3-6
Event messages	3-8
Screen descriptions	3-8
Home screen	3-9
Menu overview	3-29
Trigger menu	3-61
Scripts menu	3-63
System menu	3-66
APPS Manager	3-72
Download and run TSP applications	3-73
Examples in this manual	3-74
Display features	3-74
Setting the number of displayed digits	3-75
Setting the display format	3-76
Customizing a message for the USER swipe screen	3-77
Creating messages for interactive prompts	3-78
Save screen captures to a USB flash drive	3-79
Instrument sounds	3-80
Using the event log	3-80
Information provided for each event log entry	3-81
Event log settings	3-81
Effects of errors on scripts	3-82
Resets	3-83
Reset the instrument	3-84

Making measurements 4-1

Test connections	4-1
Basic connections	4-2
Front panel or switching module test connections	4-3
Measurement overview	4-4
Measurement capabilities	4-5
Warmup time	4-5
High-energy circuit safety precautions	4-5
DC voltage measurements	4-6
AC voltage measurements	4-10
DC current measurements	4-12
AC current measurements	4-13
Resistance measurements	4-14
Continuity measurements	4-21
Frequency measurements	4-23
Period measurements	4-24
Diode measurements	4-25
Temperature measurements	4-26
Capacitance measurements	4-30
DC voltage ratio measurements	4-31
Digitize functions	4-35
Display results of two measure functions	4-41
Displayed measurements	4-44
Using Quickset	4-44
Using the Performance slider	4-45
Store settings for functions regardless of active state	4-45
Measurement methods	4-46

Continuous measurement triggering	4-46
Trigger key triggering	4-46
Trigger model triggering	4-47
Switching between measurement methods.....	4-47
Auto Delay.....	4-48
Voltage autodelay times	4-50
Current autodelay times	4-50
Resistance autodelay times	4-51
Frequency and period autodelay times	4-51
Temperature autodelay and autorange times.....	4-51
Capacitance autodelay times	4-52
Diode autodelay times.....	4-52
Detector bandwidth	4-52
Automatic reference measurements	4-53
Setting autozero	4-53
Ranges	4-54
Selecting the automatic measurement range	4-55
Selecting a specific measure range	4-55
Relative offset	4-56
Establishing a relative offset value	4-56
Calculations that you can apply to measurements	4-59
mx+b	4-59
Percent.....	4-60
Reciprocal (1/X)	4-61
Setting mx+b math operations	4-62
Setting percent math operations	4-63
Setting reciprocal math operations.....	4-63
Switching math on the SETTINGS swipe screen	4-64
Filtering measurement data	4-65
Repeating average filter	4-66
Moving average filter	4-66
Hybrid average filter	4-67
Filter window	4-67
Setting up the averaging filter.....	4-68
Limit testing and binning	4-69
Limit testing using the front-panel interface.....	4-70
Line cycle synchronization	4-72
Using aperture or NPLCs to adjust speed and accuracy.....	4-73
DMM resistance measurement methods	4-75
Constant-current source method.....	4-76
Ratiometric method	4-77
Low-level voltage measurement considerations	4-80
Thermoelectric potentials	4-80
Magnetic fields	4-83
Radio frequency interference	4-84
Shielding	4-85
Measurement settling considerations	4-87
Reference junctions	4-88
Simulated reference junction.....	4-88
Internal reference junction.....	4-89
External reference junction.....	4-89

Order of operations	4-91
Saving setups.....	4-92
Save a user setup to internal memory.....	4-93
Save a user setup to a USB flash drive.....	4-93
Copy a user setup	4-94
Delete a user setup	4-95
Recall a user setup	4-95
Define the setup used when power is turned on	4-96
Saving front-panel settings into a macro script.....	4-97
Recording a macro script	4-98
Running a macro script	4-98
Front-panel macro recording limitations	4-99
Configuration lists.....	4-99
Configuration indexes	4-100
Working with configuration lists and indexes.....	4-100
Recall a configuration index	4-105
View configuration list contents	4-105
Delete a configuration index or list	4-107
Overwrite an existing index	4-108
List the available configuration lists.....	4-108
Determine the number of indexes in a configuration list.....	4-109
Save a configuration list	4-109
Remote commands for configuration list operations	4-110
Settings stored in a measure configuration index.....	4-111
Digitize settings stored in a measure configuration list index	4-114

Switching and scanning..... 5-1

Switching modules	5-1
Pseudocards	5-2
Connections to a switching module	5-3
Identify the installed switching module	5-4
Determine module capabilities	5-5
Channel naming	5-6
Set up measurement channels	5-7
Set up measurement channels using the front panel	5-8
Set up channels using SCPI commands	5-10
Set up channels using TSP commands	5-12
Assign labels to channels.....	5-13
Add a channel delay.....	5-14
Making measurements with channels.....	5-14
Make measurements from the front panel.....	5-15
Make channel measurements using SCPI commands.....	5-15
Make channel measurements using TSP commands	5-15
Opening and closing channels.....	5-15
Operation of switching modules	5-16
Controlling channels using the front-panel CHANNEL CONTROL screen	5-20
Controlling channels using the front-panel CHANNEL swipe screen	5-23
Control channels from a remote interface	5-24
Channel status	5-24
Relay closure counts.....	5-27
Using watch channels	5-28

Scanning and triggering	5-29
Scan operation	5-30
Set up the channels for the scan	5-31
Create a scan	5-33
Set scan options	5-35
Triggering in scans	5-47
Run a scan	5-48
Stop a scan	5-48
Check the state of a scan	5-49
Using the SCAN swipe screen	5-50
Remote commands for scans	5-54
Multiple-channel operation	5-57
Multiple-channel operation using the front panel	5-59
Control multiple channels from a remote interface	5-60
Avoid corrupt measurements when operating channels independently	5-60
Dual independent multiplexers	5-61
7700 dual multiplexer application	5-62
Reading buffers	6-1
Introduction to reading buffers	6-1
Getting started with buffers	6-2
Types of reading buffers	6-2
Effects of reset, scan start, and power cycle on buffers	6-2
Buffer fill status	6-3
Timestamps	6-5
Creating buffers	6-6
Setting reading buffer options	6-11
Selecting a buffer	6-17
Viewing and saving buffer content	6-21
Clearing buffers	6-27
Deleting buffers	6-29
Remote buffer operation	6-30
Storing data in buffers	6-31
Accessing the data in buffers	6-33
Buffer read-only attributes	6-34
Reading buffer time and date values	6-35
Reading buffer for . . . do loops	6-35
Writable reading buffers	6-36
Apply mathematical expressions to reading buffer data	6-40
Mathematical expressions for buffer math	6-41
Set up buffer math using SCPI commands	6-42
Set up buffer math using TSP commands	6-42
Using buffers across TSP-Link nodes	6-43
Graphing	7-1
Introduction	7-1
About the graph screens	7-2
How to work with the graph	7-5
Use the Graph swipe bar	7-6
Change the data that is graphed	7-9

Add, remove, and clear traces	7-10
Active buffer	7-11
Change the display of data	7-11
Change the scale of the graph	7-12
Set up triggers	7-13
Types of triggers	7-14
Trigger settings	7-14
Graph measurement using triggers	7-17
About the Histogram screen	7-18
How to work with the Histogram	7-19
Change the data that is binned	7-20
Triggering	8-1
Digital I/O	8-1
Digital I/O connector and pinouts	8-2
Digital I/O lines	8-5
Remote digital I/O commands	8-12
Digital I/O bit weighting	8-13
Digital I/O programming examples	8-13
External trigger control	8-16
Setting up the external I/O	8-17
Remote external I/O commands	8-18
Triggering	8-18
Command interface triggering	8-19
Triggering using hardware lines	8-19
Analog triggering overview	8-20
LAN triggering overview	8-24
Trigger timers	8-26
Event blenders	8-31
Interactive triggering	8-32
Trigger model	8-35
TriggerFlow Trigger Model	8-36
Trigger-model blocks	8-37
Trigger-model templates	8-59
Assembling trigger-model blocks	8-62
Running the trigger model	8-66
Using trigger events to start actions in the trigger model	8-69
TSP-Link and TSP-Net	9-1
TSP-Link System Expansion Interface	9-1
TSP-Link connections	9-2
TSP-Link nodes	9-3
Master and subordinates	9-5
Initializing the TSP-Link system	9-6
Sending commands to TSP-Link nodes	9-7
Using the reset() command	9-7
Terminating scripts on the TSP-Link system	9-8
Triggering using TSP-Link trigger lines	9-8
Running simultaneous test scripts	9-9
Using DAQ6510 TSP-Link commands with other TSP-Link products	9-16
TSP-Net	9-18

Using TSP-Net with any ethernet-enabled instrument	9-18
Remote instrument events	9-21
TSP-Net instrument commands: General device control	9-21
TSP-Net instrument commands: TSP-enabled device control	9-21
Example: Using tspnet commands	9-22
Maintenance	10-1
Introduction	10-1
Line fuse replacement.....	10-1
Line voltage verification.....	10-3
Input fuse replacement.....	10-4
Lithium battery	10-5
Front-panel display.....	10-5
Cleaning the front-panel display.....	10-6
Abnormal display operation.....	10-6
Removing ghost images or contrast irregularities	10-6
Upgrading the firmware	10-7
From the front panel.....	10-8
Using SCPI.....	10-9
Using TSP	10-10
Using TSB	10-11
Introduction to SCPI commands.....	11-1
Introduction to SCPI	11-1
Command execution rules.....	11-1
Command messages	11-2
SCPI command formatting	11-4
SCPI command short and long forms	11-4
Optional command words.....	11-4
MINimum, MAXimum, and DEFault	11-4
Queries	11-5
SCPI parameters.....	11-5
Sending strings	11-6
Channel naming	11-6
Using the SCPI command reference	11-7
Command name and summary table	11-8
Command usage.....	11-10
Command details	11-11
Example section.....	11-11
Related commands list.....	11-11
SCPI command reference.....	12-1
*RCL	12-1
*SAV	12-2
:FETCh?.....	12-3
:MEASure:DIGitize?	12-6
:MEASure?.....	12-8
:READ:DIGitize?	12-11
:READ?	12-14
CALCulate subsystem.....	12-17

:CALCulate2:<function>:LIMIT<Y>:AUDible	12-17
:CALCulate2:<function>:LIMIT<Y>:CLEar:AUTO	12-19
:CALCulate2:<function>:LIMIT<Y>:CLEar[:IMMediate]	12-20
:CALCulate2:<function>:LIMIT<Y>:FAIL?	12-22
:CALCulate2:<function>:LIMIT<Y>:LOWer[:DATA]	12-24
:CALCulate2:<function>:LIMIT<Y>:STATe	12-26
:CALCulate2:<function>:LIMIT<Y>:UPPer[:DATA]	12-27
:CALCulate[1]:<function>:MATH:FORMat	12-29
:CALCulate[1]:<function>:MATH:MBFactor	12-31
:CALCulate[1]:<function>:MATH:MMFactor	12-32
:CALCulate[1]:<function>:MATH:PERCent	12-34
:CALCulate[1]:<function>:MATH:STATe	12-36
CALibration subsystem	12-37
:CALibration:PROTeCted:CARD1:COUNT?	12-37
:CALibration:PROTeCted:CARD1:DAC:STEP0	12-37
:CALibration:PROTeCted:CARD1:DATE?	12-38
:CALibration:PROTeCted:CARD1:LOCK	12-39
:CALibration:PROTeCted:CARD1:RCount	12-40
:CALibration:PROTeCted:CARD1:REFTemp?	12-40
:CALibration:PROTeCted:CARD1:SAVE	12-41
:CALibration:PROTeCted:CARD1:STEP0	12-41
:CALibration:PROTeCted:CODE	12-42
DIGital subsystem	12-43
:DIGital:LINE<n>:MODE	12-44
:DIGital:LINE<n>:STATe	12-46
:DIGital:READ?	12-47
:DIGital:WRITe <n>	12-48
DISPlay subsystem	12-48
:DISPlay:BUFFer:ACTive	12-49
:DISPlay:CLEar	12-49
:DISPlay:<function>:DIGits	12-50
:DISPlay:LIGHt:STATe	12-51
:DISPlay:READing:FORMat	12-52
:DISPlay:SCReen	12-53
:DISPlay:USER<n>:TEXT[:DATA]	12-55
:DISPlay:WATCh:CHANnels	12-56
FORMat subsystem	12-56
:FORMat:ASCIi:PRECision	12-57
:FORMat:BORder	12-57
:FORMat[:DATA]	12-58
ROUTE subsystem	12-59
:ABORt	12-60
:INITiate[:IMMediate]	12-60
:ROUTE[:CHANnel]:CLOSE	12-61
:ROUTE[:CHANnel]:CLOSE:COUNT:INTerval	12-63
:ROUTE[:CHANnel]:CLOSE:COUNT?	12-64
:ROUTE[:CHANnel]:CONNeCt:RULE	12-65
:ROUTE[:CHANnel]:CSOhms	12-66
:ROUTE[:CHANnel]:DELay	12-68
:ROUTE[:CHANnel]:LABel	12-69
:ROUTE[:CHANnel]:MATCh	12-70
:ROUTE[:CHANnel]:MATCh:TYPE	12-71
:ROUTE[:CHANnel]:MODE	12-72
:ROUTE[:CHANnel]:MULTiple:CLOSE	12-74
:ROUTE[:CHANnel]:MULTiple:OPEN	12-75
:ROUTE[:CHANnel]:OPEN	12-76
:ROUTE[:CHANnel]:OPEN:ALL	12-77
:ROUTE[:CHANnel]:READ?	12-77

:ROUTe[:CHANnel]:STATe?	12-78
:ROUTe[:CHANnel]:TYPE?	12-79
:ROUTe[:CHANnel]:WIDTh	12-80
:ROUTe[:CHANnel]:WRITe	12-81
:ROUTe:SCAN:ADD	12-82
:ROUTe:SCAN:ADD:SINGLe	12-84
:ROUTe:SCAN:ADD:WRITe	12-85
:ROUTe:SCAN:ALARm	12-86
:ROUTe:SCAN:BUFFer	12-87
:ROUTe:SCAN:BYPass	12-88
:ROUTe:SCAN:CHANnel:STIMulus	12-89
:ROUTe:SCAN:COUNt:SCAN	12-91
:ROUTe:SCAN:COUNt:STEP?	12-92
:ROUTe:SCAN[:CREate]	12-93
:ROUTe:SCAN:EXPort	12-95
:ROUTe:SCAN:INTerval	12-97
:ROUTe:SCAN:LEARn:LIMits	12-98
:ROUTe:SCAN:MEASure:STIMulus	12-99
:ROUTe:SCAN:MEASure:INTerval	12-102
:ROUTe:SCAN:MODE	12-103
:ROUTe:SCAN:MONitor:CHANnel	12-104
:ROUTe:SCAN:MONitor:LIMit:LOWer[:DATA]	12-105
:ROUTe:SCAN:MONitor:MODE	12-106
:ROUTe:SCAN:MONitor:LIMit:UPPer[:DATA]	12-107
:ROUTe:SCAN:REStart	12-108
:ROUTe:SCAN:StARt:STIMulus	12-109
:ROUTe:SCAN:STATe?	12-111
:ROUTe:TERMinals?	12-113
SCRIpt subsystem	12-113
:SCRIpt:RUN	12-114
SENSe1 subsystem	12-115
[:SENSe[1]]:<function>:APERTure	12-116
[:SENSe[1]]:<function>:ATRigger:EDGE:LEVel	12-119
[:SENSe[1]]:<function>:ATRigger:EDGE:SLOPe	12-120
[:SENSe[1]]:<function>:ATRigger:MODE	12-121
[:SENSe[1]]:<function>:ATRigger:WINDow:DIRection	12-122
[:SENSe[1]]:<function>:ATRigger:WINDow:LEVel:HIGH	12-123
[:SENSe[1]]:<function>:ATRigger:WINDow:LEVel:LOW	12-124
[:SENSe[1]]:<function>:AVERage:COUNt	12-125
[:SENSe[1]]:<function>:AVERage[:STATe]	12-127
[:SENSe[1]]:<function>:AVERage:TCONtroll	12-129
[:SENSe[1]]:<function>:AVERage:WINDow	12-131
[:SENSe[1]]:<function>:AZERo[:STATe]	12-133
[:SENSe[1]]:<function>:BIAS:LEVel	12-134
[:SENSe[1]]:<function>:DB:REFerence	12-135
[:SENSe[1]]:<function>:DBM:REFerence	12-137
[:SENSe[1]]:<function>:DELay:AUTO	12-138
[:SENSe[1]]:<function>:DELay:USER<n>	12-139
[:SENSe[1]]:<function>:DETector:BANDwidth	12-140
[:SENSe[1]]:<function>:INPUtimpedance	12-141
[:SENSe[1]]:<function>:LINE:SYNC	12-142
[:SENSe[1]]:<function>:NPLCYcles	12-143
[:SENSe[1]]:<function>:OCOMpensated	12-145
[:SENSe[1]]:<function>:ODETector	12-147
[:SENSe[1]]:<function>:RANGe:AUTO	12-148
[:SENSe[1]]:<function>:RANGe[:UPPer]	12-149
[:SENSe[1]]:<function>:RELative	12-151
[:SENSe[1]]:<function>:RELative:ACQuire	12-153
[:SENSe[1]]:<function>:RELative:METHod	12-154
[:SENSe[1]]:<function>:RELative:STATe	12-155

[:SENSe[1]]:<function>:RTD:ALPHA	12-156
[:SENSe[1]]:<function>:RTD:BETA	12-158
[:SENSe[1]]:<function>:RTD:DELTA	12-160
[:SENSe[1]]:<function>:RTD:FOUR	12-161
[:SENSe[1]]:<function>:RTD:THREE	12-162
[:SENSe[1]]:<function>:RTD:TWO	12-163
[:SENSe[1]]:<function>:RTD:ZERO	12-164
[:SENSe[1]]:<function>:SRATE	12-165
[:SENSe[1]]:<function>:SENSe:RANGE:AUTO?	12-166
[:SENSe[1]]:<function>:SENSe:RANGE[:UPPer]?	12-167
[:SENSe[1]]:<function>:TCouple:RJUNction:SIMulated	12-168
[:SENSe[1]]:<function>:TCouple:RJUNction:RSElect	12-170
[:SENSe[1]]:<function>:TCouple:TYPE	12-171
[:SENSe[1]]:<function>:THERmistor	12-172
[:SENSe[1]]:<function>:THReshold:RANGE	12-173
[:SENSe[1]]:<function>:THReshold:RANGE:AUTO	12-174
[:SENSe[1]]:<function>:TRANSducer	12-175
[:SENSe[1]]:<function>:UNIT	12-176
[:SENSe[1]]:AZERo:ONCE	12-177
[:SENSe[1]]:CONFiguration:LIST:CATalog?	12-178
[:SENSe[1]]:CONFiguration:LIST:CREate	12-179
[:SENSe[1]]:CONFiguration:LIST:DELeTe	12-179
[:SENSe[1]]:CONFiguration:LIST:QUERy?	12-181
[:SENSe[1]]:CONFiguration:LIST:RECall	12-182
[:SENSe[1]]:CONFiguration:LIST:SIZE?	12-183
[:SENSe[1]]:CONFiguration:LIST:STORE	12-183
[:SENSe[1]]:COUNT	12-185
[:SENSe[1]]:DIGitize:COUNT	12-187
[:SENSe[1]]:DIGitize:FUNCTion[:ON]	12-188
[:SENSe[1]]:FUNCTion[:ON]	12-189
STATus subsystem	12-189
:STATus:CLEAr	12-190
:STATus:OPERation:CONDition?	12-190
:STATus:OPERation:ENABle	12-191
:STATus:OPERation:MAP	12-191
:STATus:OPERation[:EVENT]?	12-192
:STATus:PRESet	12-193
:STATus:QUEStionable:CONDition?	12-193
:STATus:QUEStionable:ENABle	12-194
:STATus:QUEStionable:MAP	12-194
:STATus:QUEStionable[:EVENT]?	12-195
SYSTem subsystem	12-196
:SYSTem:ACCEss	12-196
:SYSTem:BEEPer[:IMMediate]	12-197
:SYSTem:CARD<slot>:ACHannel[:START]?	12-198
:SYSTem:CARD<slot>:ACHannel:END?	12-199
:SYSTem:CARD<slot>:AOUTput[:START]?	12-200
:SYSTem:CARD<slot>:AOUTput:END?	12-201
:SYSTem:CARD<slot>:CSOhms?	12-202
:SYSTem:CARD<slot>:DIGitalio[:START]?	12-203
:SYSTem:CARD<X>:DIGitalio:END?	12-204
:SYSTem:CARD<slot>:IDN?	12-205
:SYSTem:CARD<slot>:ISOLated[:START]?	12-206
:SYSTem:CARD<slot>:ISOLated:END?	12-207
:SYSTem:CARD<slot>:MATRix:COLumns?	12-208
:SYSTem:CARD<slot>:MATRix[:ROWS]?	12-208
:SYSTem:CARD<slot>:TCHannel[:START]?	12-209
:SYSTem:CARD<slot>:TCHannel:END?	12-210
:SYSTem:CARD<slot>:TEMPsensor?	12-211
:SYSTem:CARD<slot>:VCHannel[:START]?	12-212

:SYSTem:CARD<slot>:VCHannel:END?	12-213
:SYSTem:CARD<X>:VMAX?	12-214
:SYSTem:CLEAr	12-214
:SYSTem:COMMunication:LAN:CONFigure	12-215
:SYSTem:COMMunication:LAN:DST:PROTection	12-216
:SYSTem:COMMunication:LAN:MACaddress?	12-217
:SYSTem:ERRor[:NEXT]?	12-218
:SYSTem:ERRor:CODE[:NEXT]?	12-219
:SYSTem:ERRor:COUNT?	12-219
:SYSTem:EVENTlog:COUNT?	12-220
:SYSTem:EVENTlog:NEXT?	12-221
:SYSTem:EVENTlog:POST	12-222
:SYSTem:EVENTlog:SAVE	12-223
:SYSTem:GPIB:ADDReSS	12-223
:SYSTem:LFRequency?	12-224
:SYSTem:PASSword:NEW	12-225
:SYSTem:PCARd<slot>	12-226
:SYSTem:POSetup	12-227
:SYSTem:TIME	12-228
:SYSTem:VERsion?	12-229
TRACe subsystem	12-229
:TRACe:ACTual?	12-229
:TRACe:ACTual:END?	12-230
:TRACe:ACTual:START?	12-231
:TRACe:CHANnel:MATH	12-233
:TRACe:CLEAr	12-236
:TRACe:DATA?	12-237
:TRACe:DELeTe	12-240
:TRACe:FILL:MODE	12-240
:TRACe:LOG:STATe	12-241
:TRACe:MAKE	12-243
:TRACe:MATH	12-245
:TRACe:POINts	12-248
:TRACe:SAVE	12-249
:TRACe:SAVE:APPend	12-251
:TRACe:STATistics:AVERAge?	12-253
:TRACe:STATistics:CLEAr	12-254
:TRACe:STATistics:MAXimum?	12-255
:TRACe:STATistics:MINimum?	12-256
:TRACe:STATistics:PK2Pk?	12-257
:TRACe:STATistics:SPAN?	12-258
:TRACe:STATistics:STDDev?	12-259
:TRACe:TRIGger	12-260
:TRACe:TRIGger:DIGitize	12-261
:TRACe:UNIT	12-262
:TRACe:WRITe:FORMat	12-264
:TRACe:WRITe:READIng	12-266
TRIGGer subsystem	12-268
:ABORt	12-269
:INITiate[:IMMediate]	12-269
:TRIGger:BLENder<n>:CLEAr	12-270
:TRIGger:BLENder<n>:MODE	12-271
:TRIGger:BLENder<n>:OVERrun?	12-271
:TRIGger:BLENder<n>:STIMulus<m>	12-272
:TRIGger:BLOCK:BRANch:ALWays	12-274
:TRIGger:BLOCK:BRANch:COUNter	12-274
:TRIGger:BLOCK:BRANch:COUNter:COUNT?	12-275
:TRIGger:BLOCK:BRANch:COUNter:RESet	12-276
:TRIGger:BLOCK:BRANch:DELTA	12-277
:TRIGger:BLOCK:BRANch:EVENT	12-278

:TRIGger:BLOCK:BRANch:LIMit:CONStant	12-281
:TRIGger:BLOCK:BRANch:LIMit:DYNamic	12-283
:TRIGger:BLOCK:BRANch:ONCE	12-285
:TRIGger:BLOCK:BRANch:ONCE:EXCLuded	12-286
:TRIGger:BLOCK:BUFFer:CLEar	12-287
:TRIGger:BLOCK:CONFig:NEXT	12-288
:TRIGger:BLOCK:CONFig:PREVious	12-289
:TRIGger:BLOCK:CONFig:RECall	12-290
:TRIGger:BLOCK:DELAy:CONStant	12-291
:TRIGger:BLOCK:DELAy:DYNamic	12-292
:TRIGger:BLOCK:DIGital:IO	12-293
:TRIGger:BLOCK:LIST?	12-294
:TRIGger:BLOCK:LOG:EVENT	12-294
:TRIGger:BLOCK:MDIGitize	12-295
:TRIGger:BLOCK:NOP	12-298
:TRIGger:BLOCK:NOTify	12-298
:TRIGger:BLOCK:WAIT	12-300
:TRIGger:CONTinuous	12-302
:TRIGger:DIGital<n>:IN:CLEar	12-303
:TRIGger:DIGital<n>:IN:EDGE	12-303
:TRIGger:DIGital<n>:IN:OVERrun?	12-304
:TRIGger:DIGital<n>:OUT:LOGic	12-305
:TRIGger:DIGital<n>:OUT:PULSewidth	12-306
:TRIGger:DIGital<n>:OUT:STIMulus	12-306
:TRIGger:EXTernal:IN:CLEar	12-308
:TRIGger:EXTernal:IN:EDGE	12-309
:TRIGger:EXTernal:IN:OVERrun?	12-310
:TRIGger:EXTernal:OUT:LOGic	12-311
:TRIGger:EXTernal:OUT:STIMulus	12-312
:TRIGger:LAN<n>:IN:CLEar	12-314
:TRIGger:LAN<n>:IN:EDGE	12-315
:TRIGger:LAN<n>:IN:OVERrun?	12-316
:TRIGger:LAN<n>:OUT:CONNeCT:STATe	12-317
:TRIGger:LAN<n>:OUT:IP:ADDReSS	12-317
:TRIGger:LAN<n>:OUT:LOGic	12-318
:TRIGger:LAN<n>:OUT:PROTOcol	12-319
:TRIGger:LAN<n>:OUT:STIMulus	12-320
:TRIGger:LOAD "ConfigList"	12-322
:TRIGger:LOAD "DurationLoop"	12-323
:TRIGger:LOAD "Empty"	12-324
:TRIGger:LOAD "GradeBinning"	12-325
:TRIGger:LOAD "LogicTrigger"	12-327
:TRIGger:LOAD "LoopUntilEvent"	12-328
:TRIGger:LOAD "SimpleLoop"	12-330
:TRIGger:LOAD "SortBinning"	12-331
:TRIGger:PAUSE	12-333
:TRIGger:RESume	12-333
:TRIGger:STATe?	12-334
:TRIGger:TIMer<n>:CLEar	12-335
:TRIGger:TIMer<n>:COUNT	12-335
:TRIGger:TIMer<n>:DELAy	12-337
:TRIGger:TIMer<n>:STARt:FRActional	12-337
:TRIGger:TIMer<n>:STARt:GENerate	12-338
:TRIGger:TIMer<n>:STARt:OVERrun?	12-339
:TRIGger:TIMer<n>:STARt:SECOnds	12-339
:TRIGger:TIMer<n>:STARt:STIMulus	12-340
:TRIGger:TIMer<n>:STATe	12-342

Introduction to TSP commands 13-1

Introduction to TSP operation	13-1
Controlling the instrument by sending individual command messages	13-2
Queries	13-4
USB flash drive path	13-4
Information on scripting and programming	13-4
Fundamentals of scripting for TSP	13-4
What is a script?.....	13-5
Runtime and nonvolatile memory storage of scripts.....	13-6
What can be included in scripts?.....	13-6
Working with scripts	13-7
Fundamentals of programming for TSP.....	13-16
What is Lua?	13-16
Lua basics	13-16
Standard libraries	13-34
Test Script Builder.....	13-38
Installing the TSB software.....	13-38
Using Test Script Builder (TSB)	13-39
Project navigator	13-40
Script editor	13-41
Outline view.....	13-41
Programming interaction	13-41
Connecting an instrument in TSB.....	13-42
Creating a new TSP project	13-43
Adding a new TSP file to a project	13-44
Running a script	13-44
Creating a run configuration.....	13-45
Memory considerations for the runtime environment.....	13-50

TSP command reference..... 14-1

TSP command programming notes	14-1
TSP syntax rules	14-1
Time and date values.....	14-4
Local and remote control.....	14-4
Using the TSP command reference.....	14-5
Command name, brief description, and summary table	14-6
Command usage.....	14-7
Command details	14-8
Example section.....	14-9
Related commands and information.....	14-9
TSP commands.....	14-10
available().....	14-10
beeper.beep().....	14-11
buffer.channelmath()	14-12
buffer.clearstats()	14-16
buffer.delete()	14-17
buffer.getstats()	14-18
buffer.make()	14-22
buffer.math().....	14-23
buffer.save()	14-27
buffer.saveappend()	14-29
buffer.unit().....	14-32
buffer.write.format()	14-34

buffer.write.reading()	14-37
bufferVar.capacity	14-40
bufferVar.channels	14-42
bufferVar.clear()	14-43
bufferVar.dates	14-44
bufferVar.endindex	14-45
bufferVar.extraformattedvalues	14-46
bufferVar.extravalues	14-48
bufferVar.extravalueunits	14-49
bufferVar.fillmode	14-51
bufferVar.formattedreadings	14-52
bufferVar.fractionalseconds	14-53
bufferVar.logstate	14-54
bufferVar.n	14-55
bufferVar.readings	14-57
bufferVar.relativetimestamps	14-58
bufferVar.seconds	14-59
bufferVar.startindex	14-60
bufferVar.statuses	14-61
bufferVar.times	14-62
bufferVar.timestamps	14-63
bufferVar.units	14-65
channel.close()	14-66
channel.connectrule	14-69
channel.getclose()	14-70
channel.getcommonside()	14-71
channel.getcount()	14-72
channel.getcountinterval()	14-73
channel.getdelay()	14-74
channel.getdmm()	14-75
channel.getlabel()	14-76
channel.getmatch()	14-76
channel.getmatchtype()	14-77
channel.getmode()	14-78
channel.getstate()	14-79
channel.gettype()	14-80
channel.getwidth()	14-81
channel.multiple.close()	14-82
channel.multiple.open()	14-82
channel.open()	14-83
channel.read()	14-84
channel.setcommonside()	14-84
channel.setcountinterval()	14-86
channel.setdelay()	14-87
channel.setdmm()	14-88
channel.setlabel()	14-99
channel.setmatch()	14-100
channel.setmatchtype()	14-100
channel.setmode()	14-102
channel.setwidth()	14-103
channel.write()	14-104
createconfigscript()	14-105
dataqueue.add()	14-106
dataqueue.CAPACITY	14-107
dataqueue.clear()	14-108
dataqueue.count	14-109
dataqueue.next()	14-110
delay()	14-112
digio.line[N].mode	14-112
digio.line[N].reset()	14-114
digio.line[N].state	14-116

digio.readport()	14-116
digio.writeport()	14-117
display.activebuffer	14-118
display.changescreen()	14-120
display.clear()	14-121
display.delete()	14-122
display.input.number()	14-123
display.input.option()	14-125
display.input.prompt()	14-127
display.input.string()	14-128
display.lightstate	14-130
display.prompt()	14-131
display.readingformat	14-133
display.settext()	14-133
display.waitevent()	14-135
display.watchchannels	14-136
dmm.digitize.analogtrigger.edge.level	14-137
dmm.digitize.analogtrigger.edge.slope	14-138
dmm.digitize.analogtrigger.mode	14-140
dmm.digitize.analogtrigger.window.direction	14-142
dmm.digitize.analogtrigger.window.levelhigh	14-144
dmm.digitize.analogtrigger.window.levellow	14-146
dmm.digitize.aperture	14-148
dmm.digitize.count	14-150
dmm.digitize.dbreference	14-152
dmm.digitize.dbmreference	14-153
dmm.digitize.displaydigits	14-154
dmm.digitize.func	14-155
dmm.digitize.inputimpedance	14-157
dmm.digitize.limit[Y].audible	14-158
dmm.digitize.limit[Y].autoclear	14-159
dmm.digitize.limit[Y].clear()	14-160
dmm.digitize.limit[Y].enable	14-162
dmm.digitize.limit[Y].fail	14-163
dmm.digitize.limit[Y].high.value	14-164
dmm.digitize.limit[Y].low.value	14-165
dmm.digitize.math.enable	14-169
dmm.digitize.math.format	14-172
dmm.digitize.math.mxb.bfactor	14-174
dmm.digitize.math.mxb.mfactor	14-176
dmm.digitize.math.percent	14-177
dmm.digitize.range	14-179
dmm.digitize.read()	14-180
dmm.digitize.readwithtime()	14-181
dmm.digitize.rel.acquire()	14-183
dmm.digitize.rel.enable	14-184
dmm.digitize.rel.level	14-185
dmm.digitize.samplerate	14-187
dmm.digitize.unit	14-189
dmm.digitize.userdelay[N]	14-190
dmm.measure.analogtrigger.edge.level	14-191
dmm.measure.analogtrigger.edge.slope	14-193
dmm.measure.analogtrigger.mode	14-195
dmm.measure.analogtrigger.window.direction	14-197
dmm.measure.analogtrigger.window.levelhigh	14-199
dmm.measure.analogtrigger.window.levellow	14-201
dmm.measure.aperture	14-203
dmm.measure.autodelay	14-205
dmm.measure.autozero.enable	14-207
dmm.measure.autorange	14-209
dmm.measure.autozero.once()	14-211

dmm.measure.bias.level	14-212
dmm.measure.configlist.catalog()	14-213
dmm.measure.configlist.create()	14-214
dmm.measure.configlist.delete()	14-215
dmm.measure.configlist.query()	14-216
dmm.measure.configlist.recall()	14-217
dmm.measure.configlist.size()	14-218
dmm.measure.configlist.store()	14-218
dmm.measure.configlist.storefunc()	14-220
dmm.measure.count	14-221
dmm.measure.dbreference	14-223
dmm.measure.dbmreference	14-225
dmm.measure.detectorbandwidth	14-227
dmm.measure.displaydigits	14-228
dmm.measure.filter.count	14-230
dmm.measure.filter.enable	14-232
dmm.measure.filter.type	14-234
dmm.measure.filter.window	14-236
dmm.measure.fourrtd	14-238
dmm.measure.func	14-240
dmm.measure.getattribute()	14-242
dmm.measure.inputimpedance	14-243
dmm.measure.limit[Y].audible	14-245
dmm.measure.limit[Y].autoclear	14-246
dmm.measure.limit[Y].clear()	14-247
dmm.measure.limit[Y].enable	14-248
dmm.measure.limit[Y].fail	14-250
dmm.measure.limit[Y].high.value	14-251
dmm.measure.limit[Y].low.value	14-252
dmm.measure.linesync	14-257
dmm.measure.math.enable	14-258
dmm.measure.math.format	14-261
dmm.measure.math.mxb.bfactor	14-263
dmm.measure.math.mxb.mfactor	14-265
dmm.measure.math.percent	14-267
dmm.measure.nplc	14-269
dmm.measure.offsetcompensation.enable	14-271
dmm.measure.opendetector	14-273
dmm.measure.range	14-275
dmm.measure.read()	14-278
dmm.measure.readwithtime()	14-279
dmm.measure.refjunction	14-280
dmm.measure.rel.acquire()	14-282
dmm.measure.rel.enable	14-283
dmm.measure.rel.level	14-284
dmm.measure.rel.method	14-287
dmm.measure.rtdalpha	14-289
dmm.measure.rtdbeta	14-291
dmm.measure.rtddelta	14-293
dmm.measure.rtdzero	14-295
dmm.measure.sense.autorange	14-297
dmm.measure.sense.range	14-298
dmm.measure.setattribute()	14-299
dmm.measure.simreftemperature	14-309
dmm.measure.thermistor	14-311
dmm.measure.thermocouple	14-313
dmm.measure.threertd	14-315
dmm.measure.threshold.autorange	14-317
dmm.measure.threshold.range	14-319
dmm.measure.transducer	14-320
dmm.measure.twortd	14-322

dmm.measure.unit	14-324
dmm.measure.userdelay[N]	14-326
dmm.reset()	14-327
dmm.terminals	14-327
eventlog.clear()	14-328
eventlog.getcount()	14-328
eventlog.next()	14-330
eventlog.post()	14-331
eventlog.save()	14-333
exit()	14-334
file.close()	14-334
file.flush()	14-335
file.mkdir()	14-337
file.open()	14-337
file.read()	14-338
file.usbdriveexists()	14-339
file.write()	14-340
format.asciiprecision	14-341
format.byteorder	14-342
format.data	14-343
fs.chdir()	14-344
fs.cwd()	14-345
fs.is_dir()	14-346
fs.is_file()	14-347
fs.mkdir()	14-348
fs.readdir()	14-349
fs.rmdir()	14-350
gpib.address	14-350
lan.dstprotection	14-351
lan.ipconfig()	14-353
lan.lxidomain	14-354
lan.macaddress	14-354
localnode.access	14-355
localnode.gettime()	14-356
localnode.gettimewithfractional()	14-356
localnode.linefreq	14-357
localnode.model	14-357
localnode.password	14-358
localnode.prompts	14-358
localnode.prompts4882	14-360
localnode.serialno	14-360
localnode.settime()	14-361
localnode.showevents	14-362
localnode.version	14-363
node[N].execute()	14-364
node[N].getglobal()	14-365
node[N].setglobal()	14-366
opc()	14-367
print()	14-368
printbuffer()	14-369
printnumber()	14-372
reset()	14-373
scan.add()	14-374
scan.addsinglestep()	14-376
scan.addwrite()	14-377
scan.alarminotify	14-378
scan.buffer	14-379
scan.bypass	14-380
scan.channel.stimulus	14-381
scan.create()	14-383
scan.export()	14-386

scan.learnlimits()	14-388
scan.list()	14-389
scan.measure.interval	14-390
scan.measure.stimulus	14-391
scan.mode	14-393
scan.monitor.channel	14-394
scan.monitor.limit.high.value	14-395
scan.monitor.limit.low.value	14-396
scan.monitor.mode	14-397
scan.restart	14-398
scan.sancount	14-399
scan.scaninterval	14-400
scan.state()	14-400
scan.start.stimulus	14-401
scan.stepcount	14-403
script.catalog()	14-404
script.delete()	14-405
script.load()	14-406
scriptVar.run()	14-407
scriptVar.save()	14-408
scriptVar.source	14-409
slot[slot].amps.endchannel	14-409
slot[slot].amps.startchannel	14-410
slot[slot].analogoutput.endchannel	14-411
slot[slot].analogoutput.startchannel	14-411
slot[slot].commonsideohms	14-412
slot[slot].digitalio.endchannel	14-413
slot[slot].digitalio.startchannel	14-413
slot[slot].idn	14-415
slot[slot].isolated.endchannel	14-416
slot[slot].isolated.startchannel	14-416
slot[slot].matrix.columns	14-417
slot[slot].matrix.rows	14-418
slot[slot].maxvoltage	14-419
slot[slot].pseudocard	14-419
slot[slot].tempsensor	14-421
slot[slot].totalizer.endchannel	14-421
slot[slot].totalizer.startchannel	14-422
slot[slot].voltage.endchannel	14-423
slot[slot].voltage.startchannel	14-423
status.clear()	14-424
status.condition	14-424
status.operation.condition	14-426
status.operation.enable	14-427
status.operation.event	14-428
status.operation.getmap()	14-429
status.operation.setmap()	14-430
status.preset()	14-431
status.questionable.condition	14-431
status.questionable.enable	14-432
status.questionable.event	14-432
status.questionable.getmap()	14-433
status.questionable.setmap()	14-434
status.request_enable	14-435
status.standard.enable	14-436
status.standard.event	14-438
timer.cleartime()	14-439
timer.gettime()	14-440
trigger.blender[N].clear()	14-440
trigger.blender[N].orenable	14-441
trigger.blender[N].overrun	14-442

trigger.blender[N].reset()	14-443
trigger.blender[N].stimulus[M]	14-444
trigger.blender[N].wait()	14-446
trigger.clear()	14-447
trigger.continuous	14-448
trigger.digin[N].clear()	14-448
trigger.digin[N].edge	14-449
trigger.digin[N].overrun	14-450
trigger.digin[N].wait()	14-451
trigger.digout[N].assert()	14-452
trigger.digout[N].logic	14-453
trigger.digout[N].pulsewidth	14-454
trigger.digout[N].release()	14-454
trigger.digout[N].stimulus	14-455
trigger.ext.reset()	14-458
trigger.extin.clear()	14-459
trigger.extin.edge	14-460
trigger.extin.overrun	14-460
trigger.extin.wait()	14-461
trigger.extout.assert()	14-462
trigger.extout.logic	14-462
trigger.extout.stimulus	14-463
trigger.lanin[N].clear()	14-465
trigger.lanin[N].edge	14-466
trigger.lanin[N].overrun	14-466
trigger.lanin[N].wait()	14-467
trigger.lanout[N].assert()	14-468
trigger.lanout[N].connect()	14-469
trigger.lanout[N].connected	14-470
trigger.lanout[N].disconnect()	14-471
trigger.lanout[N].ipaddress	14-471
trigger.lanout[N].logic	14-472
trigger.lanout[N].protocol	14-473
trigger.lanout[N].stimulus	14-474
trigger.model.abort()	14-476
trigger.model.getblocklist()	14-476
trigger.model.getbranchcount()	14-477
trigger.model.initiate()	14-478
trigger.model.load() — ConfigList	14-478
trigger.model.load() — DurationLoop	14-480
trigger.model.load() — Empty	14-481
trigger.model.load() — GradeBinning	14-481
trigger.model.load() — LogicTrigger	14-484
trigger.model.load() — LoopUntilEvent	14-485
trigger.model.load() — SimpleLoop	14-487
trigger.model.load() — SortBinning	14-488
trigger.model.pause()	14-491
trigger.model.resume()	14-492
trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS	14-493
trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER	14-493
trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA	14-494
trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT	14-496
trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC	14-498
trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT	14-500
trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE	14-502
trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE_EXCLUDED	14-502
trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR	14-503
trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT	14-504
trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV	14-505
trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL	14-506
trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT	14-507

trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC	14-508
trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO	14-509
trigger.model.setblock() — trigger.BLOCK_LOG_EVENT	14-510
trigger.model.setblock() — trigger.BLOCK_MEASURE_DIGITIZE	14-511
trigger.model.setblock() — trigger.BLOCK_NOP	14-515
trigger.model.setblock — trigger.BLOCK_NOTIFY	14-516
trigger.model.setblock() — trigger.BLOCK_RESET_BRANCH_COUNT	14-517
trigger.model.setblock() — trigger.BLOCK_WAIT	14-518
trigger.model.state()	14-520
trigger.timer[N].clear()	14-521
trigger.timer[N].count	14-522
trigger.timer[N].delay	14-523
trigger.timer[N].delaylist	14-524
trigger.timer[N].enable	14-525
trigger.timer[N].reset()	14-526
trigger.timer[N].start.fractionalseconds	14-527
trigger.timer[N].start.generate	14-528
trigger.timer[N].start.overrun	14-529
trigger.timer[N].start.seconds	14-530
trigger.timer[N].start.stimulus	14-530
trigger.timer[N].wait()	14-533
trigger.tsplinkin[N].clear()	14-534
trigger.tsplinkin[N].edge	14-535
trigger.tsplinkin[N].overrun	14-536
trigger.tsplinkin[N].wait()	14-536
trigger.tsplinkout[N].assert()	14-537
trigger.tsplinkout[N].logic	14-538
trigger.tsplinkout[N].pulsewidth	14-538
trigger.tsplinkout[N].release()	14-539
trigger.tsplinkout[N].stimulus	14-540
trigger.wait()	14-542
tsplink.group	14-543
tsplink.initialize()	14-543
tsplink.line[N].mode	14-545
tsplink.line[N].reset()	14-545
tsplink.line[N].state	14-546
tsplink.master	14-547
tsplink.node	14-548
tsplink.readport()	14-549
tsplink.state	14-550
tsplink.writeport()	14-551
tspnet.clear()	14-552
tspnet.connect()	14-553
tspnet.disconnect()	14-555
tspnet.execute()	14-556
tspnet.idn()	14-557
tspnet.read()	14-558
tspnet.readavailable()	14-559
tspnet.reset()	14-560
tspnet.termination()	14-561
tspnet.timeout	14-562
tspnet.tsp.abort()	14-563
tspnet.tsp.abortonconnect	14-564
tspnet.tsp.rbttablecopy()	14-565
tspnet.tsp.runscript()	14-566
tspnet.write()	14-567
upgrade.previous()	14-567
upgrade.unit()	14-568
userstring.add()	14-569
userstring.catalog()	14-570
userstring.delete()	14-571

userstring.get()	14-571
waitcomplete()	14-572

Common commands 15-1

Introduction	15-1
*CLS	15-2
*ESE	15-3
*ESR?	15-5
*IDN?	15-6
*LANG	15-8
*OPC	15-9
*RST	15-9
*SRE	15-10
*STB?	15-11
*TRG	15-11
*TST?	15-12
*WAI	15-12

Status model 16-1

Overview	16-1
Standard Event Register	16-3
Programmable status register sets	16-5
Status Byte Register	16-11
Queues	16-14
Serial polling and SRQ	16-15
Programming enable registers	16-16
Reading the registers	16-17
Understanding bit settings	16-18
Clearing registers	16-19
Status model programming examples	16-20
SRQ on error	16-20
SRQ when reading buffer becomes full	16-20

Frequently asked questions..... 17-1

How do I save the present state of the instrument?	17-2
What is the ethernet port number?	17-2
What to do if the GPIB controller is not recognized?	17-3
I am receiving GPIB timeout errors. What should I do?	17-3
Can I use Keysight GPIB cards with Keithley drivers?	17-3
What does –113 "Undefined header" error mean?	17-4
What does –410 "Query interrupted" error mean?	17-4
What does –420 "Query unterminated" error mean?	17-5
What is error 5503?	17-5
How do I upgrade the firmware?	17-5
Where can I find updated drivers?	17-6

Why did my settings change?	17-7
Why can't the DAQ6510 read my USB flash drive?.....	17-7
How do I check the USB driver for the device?	17-7
Why are there dashes on the front-panel display?	17-9
How do I display the instrument's serial number?	17-10
Why doesn't the DAQ6510 recognize my switch card?	17-11
Where is the current fuse located?	17-11
Where can I find the EU Declaration of Conformity?	17-11
What VISA resource name is required?.....	17-12
How do I control the backplane channels using multiple close commands?	17-12
Can you program the TERMINALS switch?.....	17-13
Are the channels on the switching module isolated from earth ground?	17-13
Getting readings on scale in four-wire ohms.	17-13
Is the thermistor sensor for temperature two-wire or four-wire?	17-13
What is offset compensation?	17-13
What are the test current values for the resistance ranges?	17-14
What are the current shunt resistors in the DAQ6510?	17-14
What happens when thermocouple open detect is enabled?	17-14
What is a configuration list?	17-15
How do I change the command set?	17-15
How do I obtain the fastest measurement rate?	17-16
Can I close one channel and keep it closed while other channels are scanned?	17-17
I see a command that is not in the manual. What is it?	17-18
Can the DAQ6510 have multiple user-defined RTDs in a scan list?	17-18
What rack kit is required for mounting?	17-18
Which rack kit is required to rack mount a DAQ6510 and 2280 side-by-side?	17-18
Next steps	18-1
Additional DAQ6510 information.....	18-1

Introduction

In this section:

Welcome	1-1
Extended warranty	1-2
Contact information	1-2
Organization of manual sections	1-3
Capabilities and features.....	1-4
General ratings.....	1-4

Welcome

The Model DAQ6510 is a precise data-acquisition and logging system that provides a new level of simplicity compared with the often-complicated configuration and control found in many stand-alone solutions. A 5-inch multi-touch display guides users through setup, data visualization, and analysis, removing the necessity of a computer and custom software for many applications. If you need a computer, a complement of IVI and LabVIEW drivers and the Kickstart start-up program are available to help with any application.

Using Keithley Instrument's newest 6.5-digit multimeter technology, you will find even greater accuracy, functionality, and speed. Two-year accuracy specifications are included so you can be assured your critical measurements remain accurate and fully traceable.

With 12 plug-ins switching and control modules available and with two module slots, you can build a test system that can measure or control up to 80 devices-under-test (DUTs) in a multiplexing configuration. With two 6 x 8 matrix modules (a total of 96 crosspoints) the DAQ6510 can test one or multiple integrated circuits. For high-volume production testing, select a solid-state multiplexer module to maximize throughput with 500 channel/s scanning speed and minimize downtime compared with fixed life, electro-mechanical relays. A number of modules have cold junction compensation, temperature references for thermocouple temperature monitoring of devices during environmental testing, and HALT and HASS accelerated life testing. You can even test wireless devices with RF modules that have a maximum switching frequency of 3.5 GHz.

Extended warranty

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

Contact information

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

Organization of manual sections

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

- **Instrument description:** Describes the front-panel interface, features, and functions.
- **General operation:** Describes the components of the instrument and basic operation.
- **Making measurements:** Describes best practices and recommended procedures that can increase measurement speed, accuracy, and sensitivity.
- **Switching and scanning:** Describes how to install, connect, and control scanner cards.
- **Reading buffers:** Describes how reading buffers provide statistics, including average, minimum, maximum, and standard deviation.
- **Graphing:** Describes how the graphing features allow you to view your measurement data graphically. You can view minimums and maximums, view averages, determine deltas, and view the values of specific data points.
- **Triggering:** Describes the triggering options, including command-interface triggering, timers, analog trigger, event blenders, and the trigger model.
- **TSP-Link and TSP-Net:** Describes TSP-Link®, a high-speed trigger synchronization and communications bus that you can use to connect multiple instruments in a master and subordinate configuration.
- **Maintenance:** Contains information about instrument maintenance, including line fuse replacement and firmware upgrades.
- **Introduction to SCPI commands:** Describes how to control the instrument using SCPI commands.
- **SCPI command reference:** Contains programming notes and an alphabetical listing of all SCPI commands available for the DAQ6510.
- **Introduction to TSP operation:** Describes the basics of using Test Script Processor (TSP®) commands to control the instrument and describes how to control the instrument using TSP commands and Test Script Builder (TSB) software, TSP-Link system expansion, and TSP-Net.
- **TSP command reference:** Contains programming notes and an alphabetical listing of all TSP commands available for the DAQ6510.
- **Common commands:** Contains descriptions of IEEE Std 488.2 common commands.
- **Status model:** Describes the DAQ6510 status model.
- **Frequently asked questions:** Contains information that answers commonly asked questions.
- **Next steps:** Contains sources of additional information.

Capabilities and features

The DAQ6510 includes the following features:

- Large five-inch multi-touch capacitive touchscreen with graphical display
- Two-year-specified, full featured, traceable 6.5-digit multimeter with 0.0035 percent DCV (10V) basic accuracy
- Standard LAN/LXI, USB device, and host communications interfaces (GPIB optional)
- Twelve different switch, RF, and control modules to connect to as many as 80 DUTs in one test setup
- Up to 80 channels of thermocouple, RTD, or thermistor temperature measurements
- Scanning speed up to 500 channels/s with solid-state relay module
- Front-panel connections for stand-alone DMM operation

General ratings

The general ratings of the DAQ6510 are listed in the following table.

Category	Specification
Supply voltage range	100 V setting: 90 V to 110 V 120 V setting: 108 V to 132 V 220 V setting: 198 V to 242 V 240 V setting: 216 V to 264 V
Supply voltage frequency	50 Hz, 60 Hz, or 400 Hz (automatically sensed at power on)
Input and output connections	See Rear panel overview (on page 3-3).
Environmental conditions	For indoor use only Altitude: Maximum 2000 meters (6562 feet) above sea level Operating: 0 °C to 50 °C, ≤80 percent relative humidity at 35 °C Storage: –40 °C to 70 °C Pollution degree: 2

In this section:

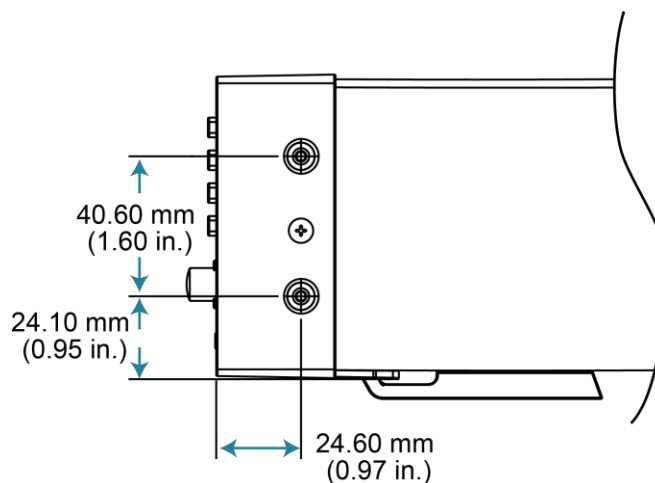
Dimensions	2-1
Bumpers and extension feet.....	2-3
Installing a switching module.....	2-4
Instrument power	2-5
Remote communications interfaces	2-7
Determining the command set you will use	2-41
Interface access	2-42
System information	2-45

Dimensions

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

The following figure shows the mounting screw locations and dimensions. Mounting screws must be #6-32 with a maximum screw length of 11.12 mm (0.438 in.) or 7/16 in. The dimensions shown are typical for both sides of the instrument.

Figure 1: DAQ6510 mounting screw locations and dimensions



The following figures show the dimensions with the bumpers installed.

Figure 2: DAQ6510 dimensions with bumpers installed

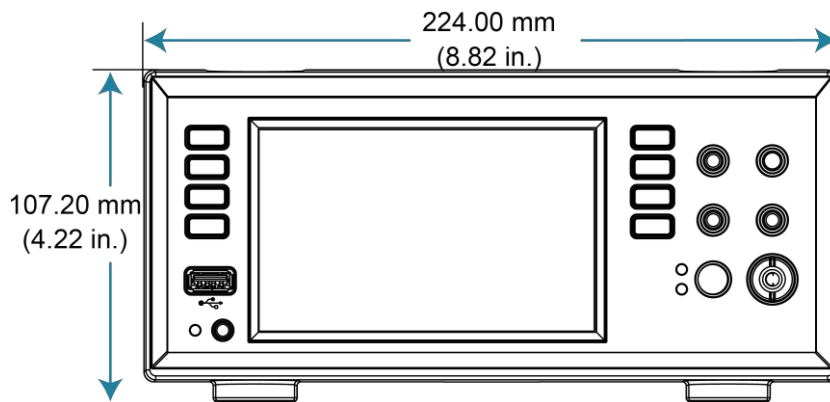
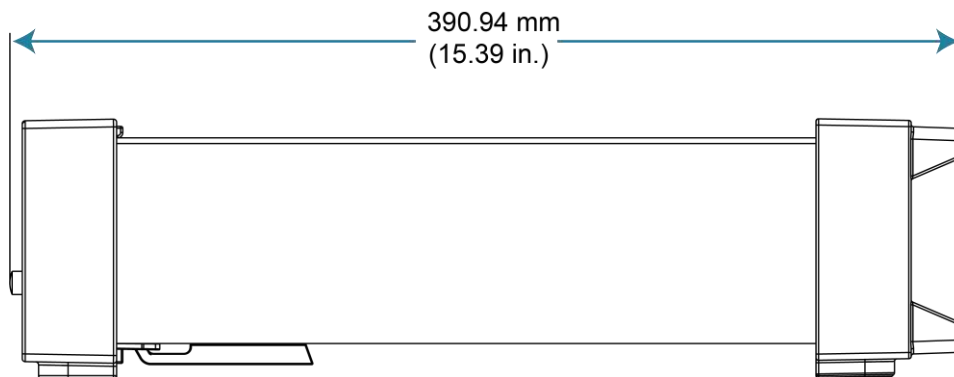


Figure 3: DAQ6510 side dimensions with bumpers installed



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

Figure 4: DAQ6510 dimensions with bumpers removed

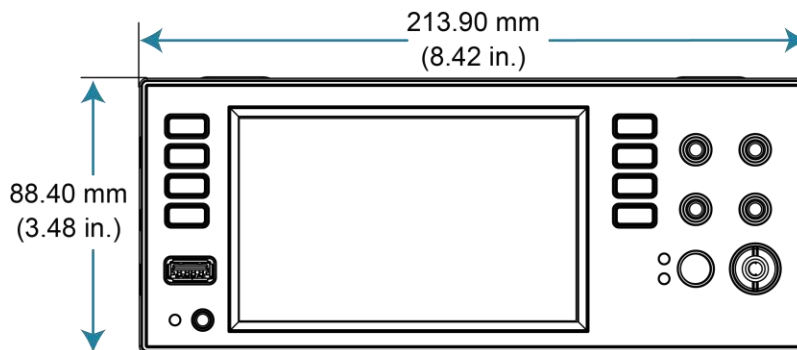
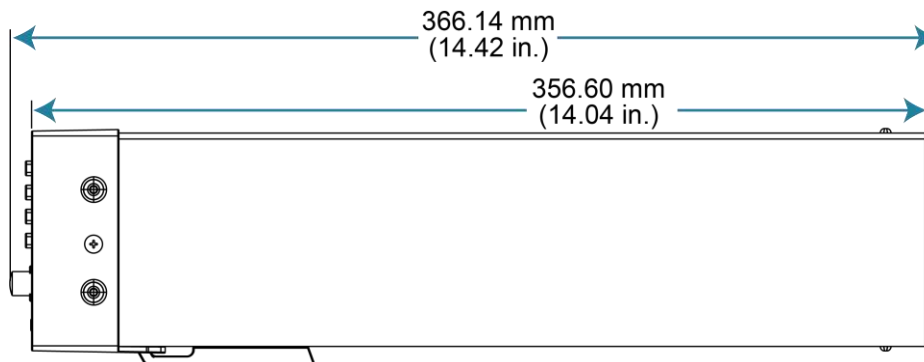


Figure 5: DAQ6510 side dimensions with bumpers removed

Bumpers and extension feet

The DAQ6510 has front and rear bumpers for using the instrument on a benchtop. The instrument also features tilt feet that you can extend or retract to improve front-panel viewing.

You can remove the bumpers and tilt feet for rack mounting.

NOTE

See tek.com/keithley (tek.com/products/keithley) for compatible rack-mount kits.

Removing the bumpers

You can remove the bumpers on the DAQ6510 if you want to mount the instrument in a rack.

NOTE

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

To remove the bumpers:

1. Remove all connections to the instrument.
2. Grasp the front bumper on each side of the DAQ6510 and gently pull it toward you until the bumper comes off the instrument.
3. Repeat for the rear bumper.

Adjusting the tilt feet

You can flip the tilt feet on the DAQ6510 for easier front-panel viewing.

Removing the tilt feet for rack mounting

You can remove the tilt feet on the DAQ6510 to mount the instrument in a rack.

NOTE

If you remove the tilt feet, be sure to store them for future benchtop use.

NOTE

Remove all connections and the power cord from the DAQ6510 before you begin.

To remove the tilt feet:

1. Squeeze the sides of one tilt foot near where it attaches to the instrument and twist the foot until it detaches.
2. Repeat the previous step for the other side of the DAQ6510.
3. Store the tilt feet for future use.

Installing a switching module

Refer to the instructions for your switching module for installation information.

Instrument power

Follow the steps below to connect the DAQ6510 to line power and turn on the instrument. The DAQ6510 operates from a line voltage of 100 V to 240 V at a frequency of 50 Hz, 60 Hz, or 400 Hz. It automatically senses line frequency. Make sure the operating voltage in your area is compatible.

The fuse is set to the expected voltage at the factory. Make sure that the correct line voltage is displayed on the power module. See [Line voltage verification](#) (on page 10-3) for more information.

NOTE

You must turn on the DAQ6510 and allow it to warm up for at least 30 minutes to achieve rated accuracies.

CAUTION

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

WARNING

The power cord supplied with the DAQ6510 contains a separate protective earth (safety ground) wire for use with grounded outlets. When proper connections are made, the instrument chassis is connected to power-line ground through the ground wire in the power cord. In the event of a failure, not using a properly grounded protective earth and grounded outlet may result in personal injury or death due to electric shock.

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

Connect the power cord

When you connect the power cord, the instrument may power on, depending on the state of the front-panel POWER switch.

To connect the power cord:

1. Connect the female end of the supplied power cord to the AC receptacle on the rear panel.
2. Connect the male end of the power cord to a grounded AC outlet.

Turn the DAQ6510 on or off

WARNING

Before installing the instrument, disconnect all external power from the equipment and disconnect the line cord. Failure to disconnect all power may expose you to hazardous voltages, which, if contacted, could cause personal injury or death.

NOTE

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

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

To turn a DAQ6510 on:

1. Disconnect any devices under test (DUTs) from the DAQ6510.
2. Press the front-panel **POWER** switch to place it in the on position.

The instrument displays a status bar as the instrument powers on. The home screen is displayed when power on is complete.

To turn a DAQ6510 off:

Press and hold the front-panel **POWER** switch to place it in the off position.

Remote communications interfaces

You can choose one of several communications interfaces to send commands to and receive responses from the DAQ6510.

The instrument automatically detects the type of communications interface (LAN, USB, GPIB, RS-232, or TSP-Link®) when you connect to the respective port on the rear panel of the instrument. The GPIB, RS-232, and TSP-Link options require an optional accessory card. In most cases, you do not need to configure anything on the instrument. In addition, you do not need to reboot if you change the type of interface that is connected.

You can only use one communications interface to control the DAQ6510 at a time. The USB connection takes precedence over LAN connections. For other communications interfaces, the first interface on which the instrument receives a message takes control of the instrument. If another interface sends a message, that interface can take control of the instrument. You may need to enter a password to change the interface, depending on the selected interface access.

Supported remote interfaces

The DAQ6510 supports the following remote interfaces:

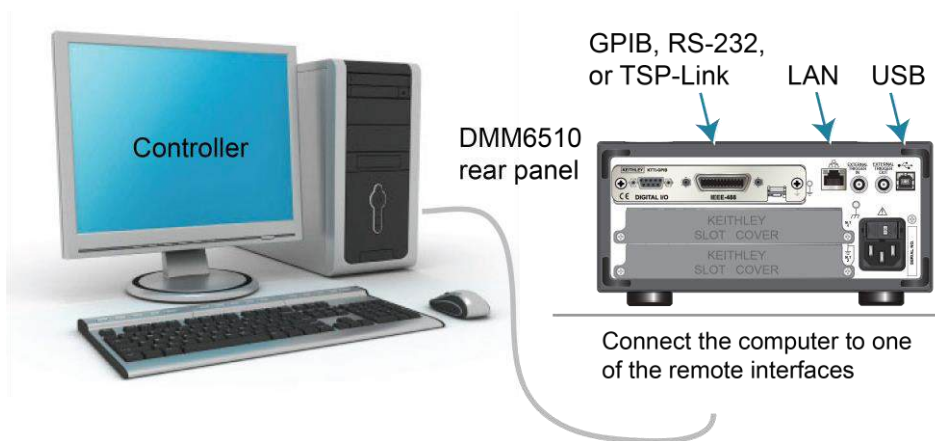
- **GPIO:** IEEE-488 instrumentation general-purpose interface bus
- **Ethernet:** Local-area-network communications
- **RS-232:** Serial communications data standard
- **USB:** Type B USB port
- **TSP-Link®:** A high-speed trigger synchronization and communications bus that test system builders can use to connect multiple instruments in a master-and-subordinate configuration. For details about TSP-Link, see [TSP-Link System Expansion Interface](#) (on page 9-1).

The DAQ6510 can be controlled from only one communications interface at a time.

NOTE

The GPIO, RS-232, and TSP-Link interfaces require an optional communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIO, KTTI-TSP, and KTTI-RS232.

Figure 6: DAQ6510 remote interface connections



Comparison of the communications interfaces

The following topics discuss some of the advantages and disadvantages of the communications interfaces that are available for the DAQ6510.

Simplicity

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

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

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

Triggering

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

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

Transfer rate

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

Instrument naming

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

Distance and instrument limitations

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

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

Expense

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

GPIB setup

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

The DAQ6510 GPIB interface is IEEE Std 488.1 compliant and supports IEEE Std 488.2 common commands and status model topology.

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

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

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

NOTE

GPIB communications require the KTTI-GPIB communications accessory card to be installed in the instrument.

Install the GPIB driver software

Check the documentation for your GPIB controller for information about where to acquire drivers. Keithley also recommends that you check with the manufacturer of the GPIB controller for the latest version of drivers or software.

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

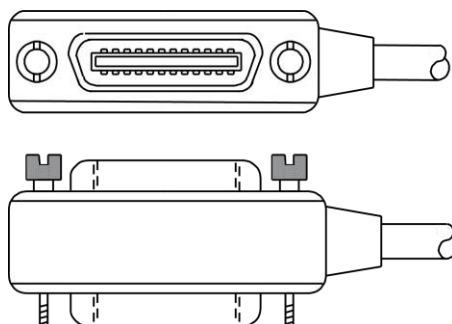
Install the GPIB cards in your computer

Refer to the documentation from the GPIB controller manufacturer for information about installing the GPIB controllers.

Connect the GPIB cables to your instrument

To connect a DAQ6510 to the GPIB interface, use a cable equipped with standard GPIB connectors, as shown in the following figure.

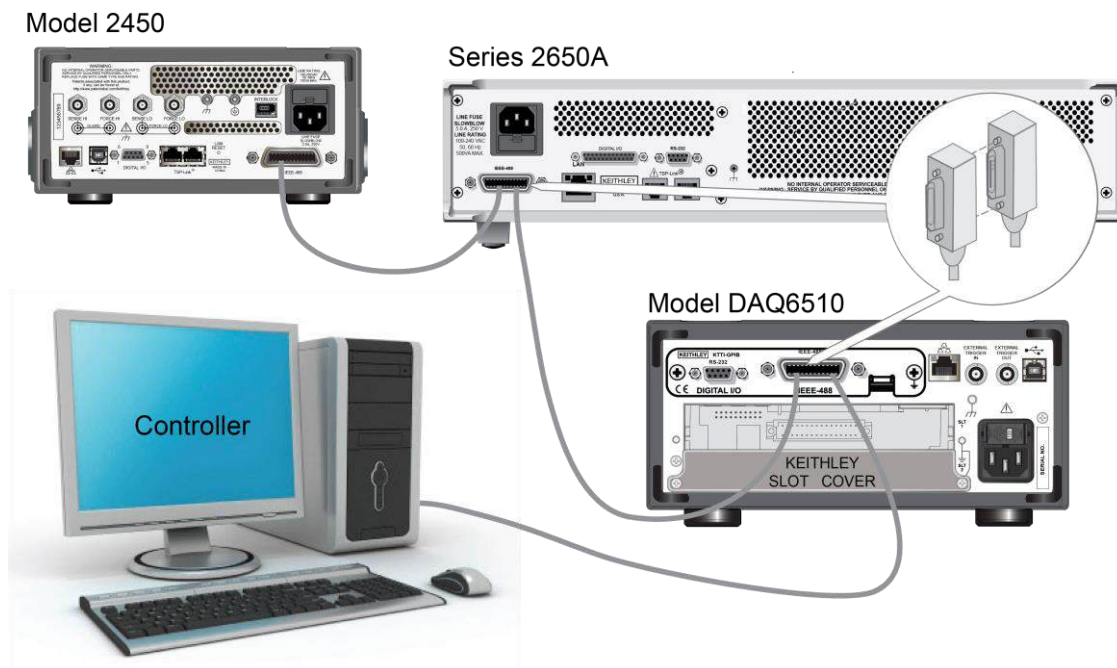
Figure 7: GPIB connector



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

CAUTION

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

Figure 8: DAQ6510 instrument GPIB connections

To connect the GPIB cable to the instrument:

1. Align the cable connector with the connector on the DAQ6510 rear panel.
2. Attach the connector. Tighten the screws securely but do not overtighten them.
3. Connect any additional connectors from other instruments, as required for your application.
4. Make sure that the end of the cable is properly connected to the controller.

Set the GPIB address

The default GPIB address is 16. You can set the address from 1 to 30 if it is unique in the system. This address cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

NOTE

GPIB controllers are usually set to 0 or 21. To be safe, do not configure any instrument to have an address of 21.

The instrument saves the address in nonvolatile memory. It does not change when you send a reset command or when you turn the power off and on again.

To set the GPIB address from the front panel:

1. Press the **MENU** key.
2. Select **Communication**.
3. Select the **GPIB** tab.
4. Set the GPIB **Address**.
5. Select **OK**.

NOTE

You can also set the GPIB address using remote commands. Set the GPIB address with the SCPI command `:SYSTem:GPIB:ADDRes` or the TSP command `gpiB.address`.

Effect of GPIB line events on the DAQ6510

The GPIB has control lines that allow predefined information, called events, to be transferred quickly. The following information lists some of the GPIB line events and how the DAQ6510 reacts to them.

DCL

This event clears the GPIB interface. When the DAQ6510 detects a device clear (DCL) event, it does the following:

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

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

GET

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

GTL

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

IFC

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

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

LLO

When the instrument detects a local-lockout (LLO) event, all front-panel controls and the POWER switch are disabled.

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

REN

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

You should place the instrument into remote mode before you attempt to program it using a remote interface.

SDC

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

When the DAQ6510 detects an SDC event, it will do the following for the selected instrument:

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

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

SPE, SPD

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

The serial poll byte contains information about internal functions. Generally, the serial polling sequence is used by the controller to determine which of several instruments has requested service with the SRQ line. See the [Status model](#) (on page 16-1) for detail.

USB communications

To use the rear-panel USB port, you need a driver that communicates using the USBTMC protocol, such as NI-VISA, on the host computer.

When installed, the USBTMC protocol allows the Microsoft® Windows® operating system to recognize the instrument.

When you connect a USB device that implements the USBTMC or USBTMC-USB488 protocol to the computer, the driver automatically detects the device. Note that the driver does not recognize other USB devices, such as printers, scanners, and storage devices.

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

The USB connection takes precedence over LAN connections. To use LAN, you must disconnect the USB connection.

See [How to install the Keithley I/O Layer](#) (on page 2-40) for more information on drivers.

Using USB

To communicate from a computer to the instrument, you need a USB cable with a USB Type B connector end and a USB Type A connector end. You need a separate USB cable for each instrument you plan to connect to the computer at the same time using the USB interface.

To connect an instrument to a computer using USB:

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

Communicate with the instrument

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

```
USB0::0x05e6::0x6510::[serial number]::INSTR
```

Where:

- 0x05e6: The Keithley vendor ID
- 0x6510: The instrument model number
- [serial number]: The serial number of the instrument (the serial number is also on the rear panel)
- INSTR: Use the USBTMC protocol

The resource string is displayed on the bottom right of the System Communications screen when the USB connection is active. Select **Menu**, then **Communication** to open the System Communications menu and select the **USB** tab.

You can also retrieve the resource string by running the Keithley Configuration Panel, which automatically detects all instruments connected to the computer.

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

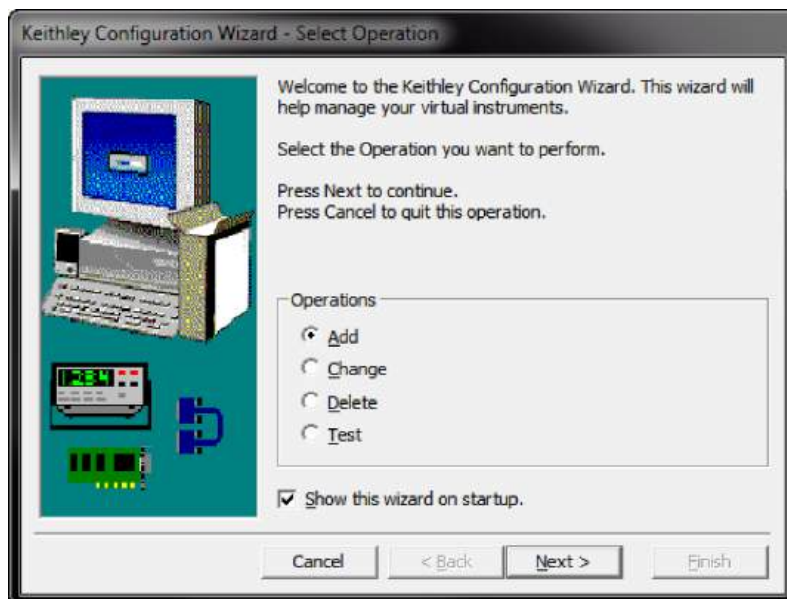
NOTE

If you have a USB connection, you cannot switch to a LAN connection while the USB is connected. USB takes precedence over LAN.

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

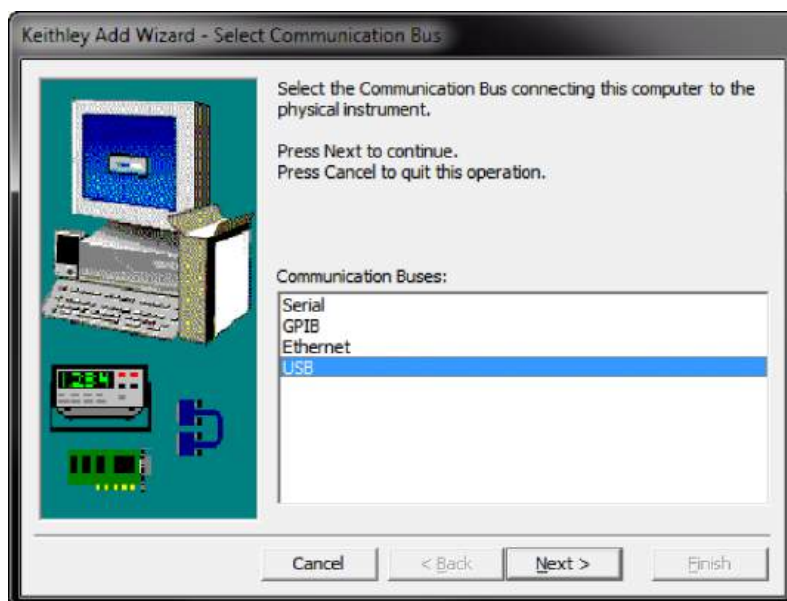
1. Select **Start > Keithley Instruments > Keithley Configuration Panel**. The Select Operation dialog box is displayed.

Figure 9: Select Operation dialog box



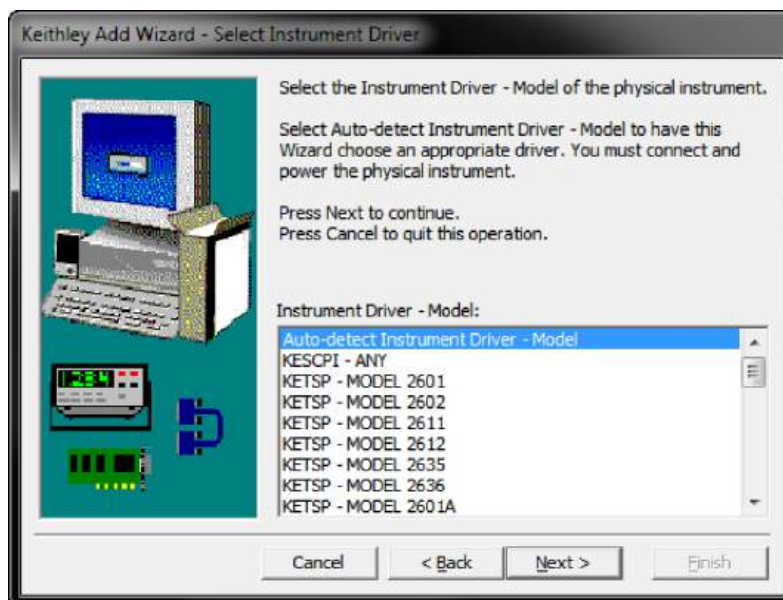
2. Select **Add**.
3. Select **Next**. The Select Communication Bus dialog box is displayed.

Figure 10: Select Communication Bus dialog box

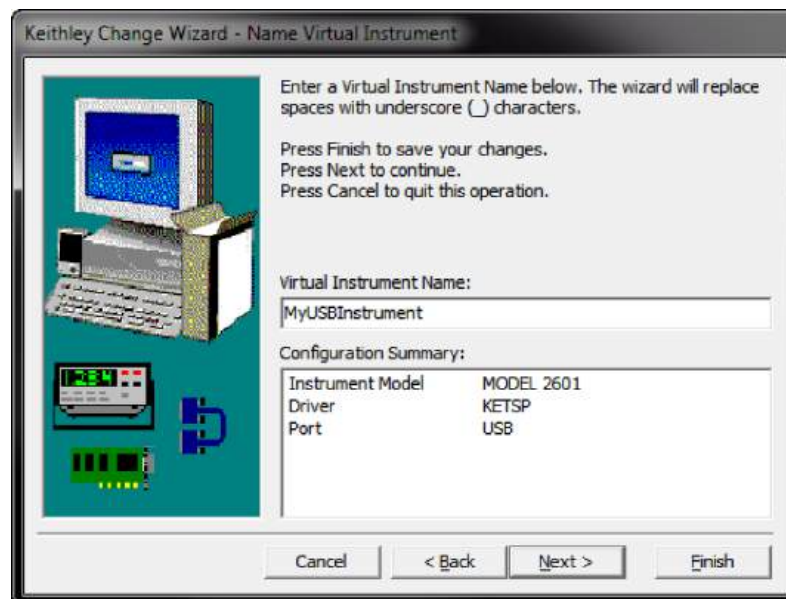


4. Select **USB**.
5. Select **Next**. The Select Instrument Driver dialog box is displayed.

Figure 11: Select Instrument Driver dialog box



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

Figure 12: Name Virtual Instrument dialog box

9. In the Virtual Instrument Name box, enter a name that you want to use to refer to the instrument.
10. Select **Finish**.
11. Select **Cancel** to close the Wizard.
12. Save the configuration. From the Keithley Configuration Panel, select **File > Save**.

Verify the instrument through the Keithley Communicator:

1. Set the instrument to use the SCPI command set. Refer to [How do I change the command set?](#) (on page 17-15) for instruction.
2. Select **Start > Keithley Instruments > Keithley Communicator**.
3. Select **File > Open Instrument** to open the instrument you just named.

Figure 13: Keithley Communicator Open an Instrument

4. Select **OK**.
5. Send a command to the instrument and see if it responds.

NOTE

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

LAN communications

You can communicate with the instrument using a local area network (LAN). The LAN interface can be used to build flexible test systems that include web access. This section provides an overview of LAN communications for the DAQ6510.

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

The DAQ6510 is a version 1.5 LXI Device Specification 2016 instrument that supports TCP/IP and complies with IEEE Std 802.3 (ethernet LAN). There is one LAN port (located on the rear panel of the instrument) that supports full connectivity on a 10 Mbps or 100 Mbps network. The DAQ6510 automatically detects the speed.

The DAQ6510 also supports Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD), which are useful on a LAN with no central administration.

NOTE

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

If you have problems setting up the LAN, refer to [LAN troubleshooting suggestions](#) (on page 2-32).

NOTE

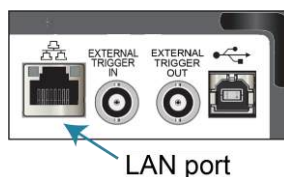
The USB connection takes precedence over LAN connections. To use LAN, you must disconnect the USB connection.

LAN cable connection

You can use any standard LAN crossover cable (RJ-45, male to male) or straight-through cable to connect your equipment. The instrument automatically senses which cable you have connected.

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

Figure 14: DAQ6510 LAN port



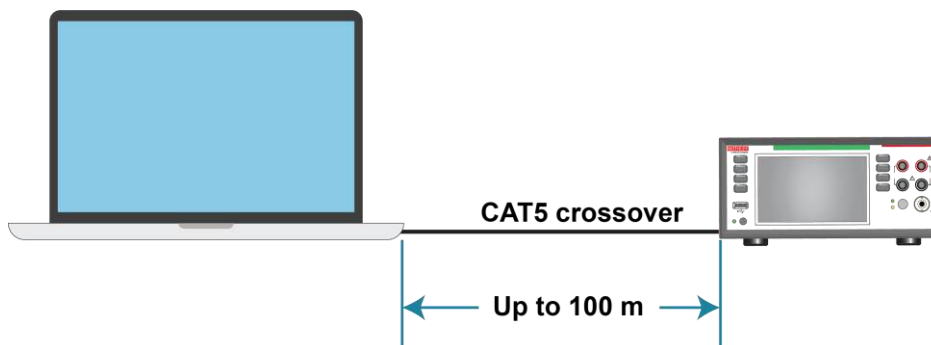
You can connect the instrument to the LAN in a one-to-one, one-to-many, two network cards, or enterprise configuration, as described in the following topics.

One-to-one connection

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

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

Figure 15: One-to-one connection with a crossover cable



NOTE

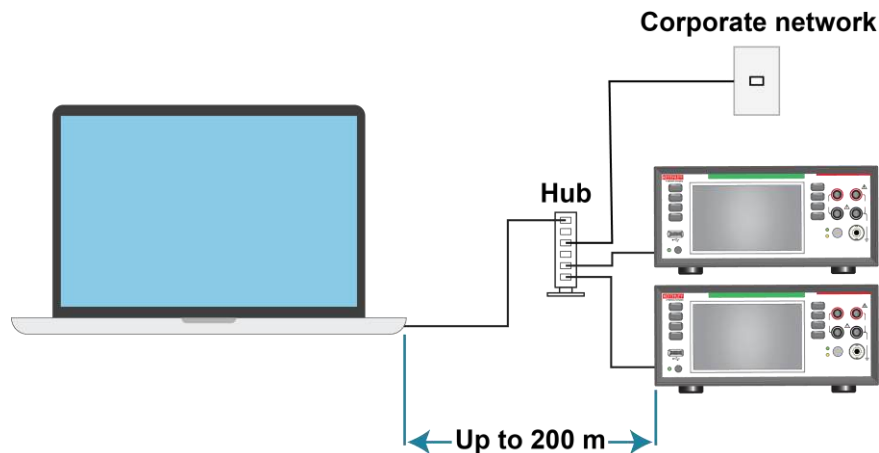
The DAQ6510 supports Auto-MDIX and can use either normal LAN CAT-5 cables (patch) or crossover cables. The instrument automatically adjusts to support either cable.

One-to-many connection

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

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

Figure 16: One-to-many connection using a network hub or switch

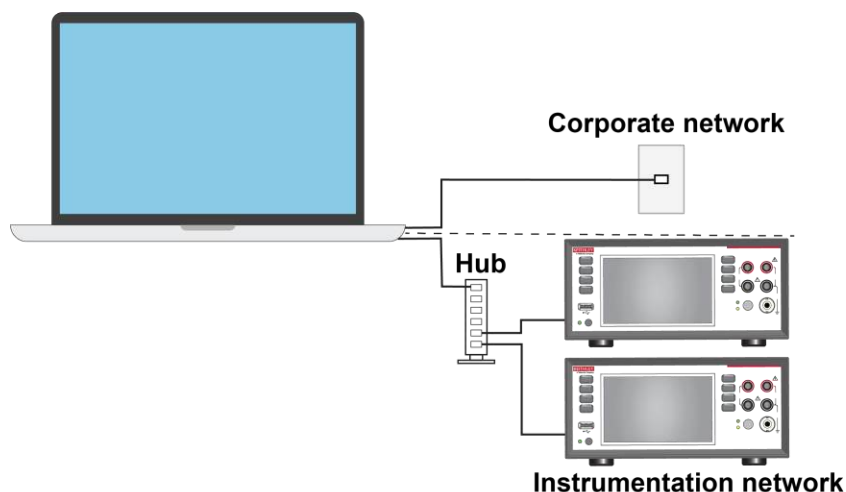


Two network card connection

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

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

Figure 17: Two network card connection

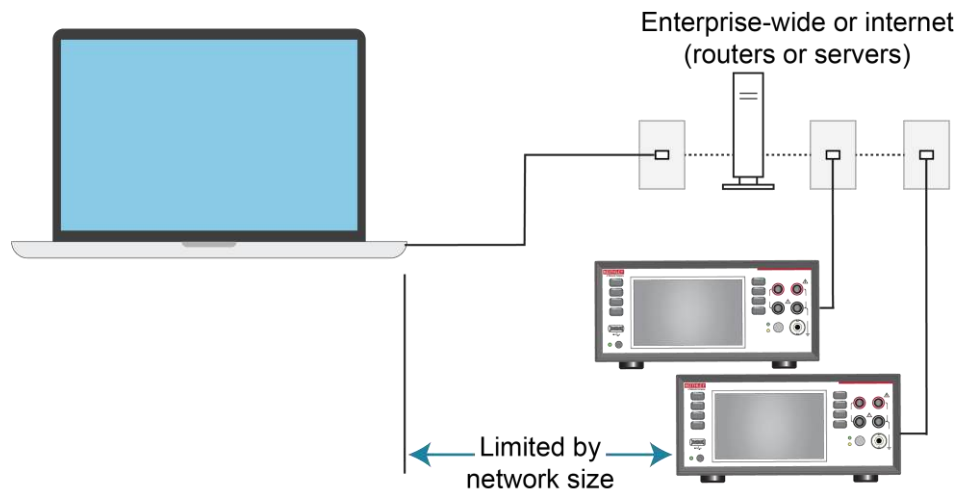


Instrumentation connection to enterprise routers or servers

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

Usually, the instruments are kept inside the corporate firewall, but the network administrator can assign resources that allow them to be outside the firewall. This allows instruments to be connected to the internet using appropriate security methods. Data collection and distribution can be controlled from virtually any location.

Figure 18: Instrumentation connection to enterprise routers or servers



Set up LAN communications on the instrument

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

Check communications settings

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

To check communications settings on the instrument:

1. Press the **MENU** key.
2. Under System, select **Communication**. The SYSTEM COMMUNICATIONS window opens.
3. Select one of the tabs (**GPIB**, **USB**, **LAN**, **RS-232**, or **TSP-Link**) to see the settings for that interface.
4. Press the **EXIT** key to leave the SYSTEM COMMUNICATIONS window without making any changes.

Set up automatic LAN configuration

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

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

NOTE

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

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

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

Set up manual LAN configuration

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

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

NOTE

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

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

To set up manual IP address selection on the instrument:

1. Press the **MENU** key.
2. Under System, select **Communication**.
3. Select the **LAN** tab.
4. For TCP/IP Mode, select **Manual**.
5. Enter the **IP Address**.
6. Enter the **Gateway** address.
7. Enter the **Subnet** mask.
8. Select **Apply Settings** to save your settings.

Set up LAN communications on the computer

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

NOTE

Do not change your IP address without consulting your system administrator. If you enter an incorrect IP address, it can prevent your computer from connecting to your corporate network or it may cause interference with another networked computer.

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

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

Verify the LAN connection on the DAQ6510

Make sure that your DAQ6510 is connected to the network by confirming that your instrument was assigned an IP address.

To verify the LAN connection:

1. Press the **MENU** key.
2. Under System, select **Communication**.
3. Select the **LAN** tab.

A green LAN status indicator on the lower left of the LAN tab confirms that your instrument was assigned an IP address.

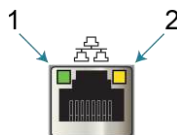
Use the LXI Discovery Tool

To find the IP address of the DAQ6510 from a computer, use the LXI Discovery Tool, a utility that is available from the Resources tab of the [LXI Consortium website \(lxistandard.org\)](http://lxistandard.org).

LAN status LEDs

The following figure shows the two status LEDs on the LAN port of the instrument. The table below the figure provides explanations of the LED states.

Figure 19: DAQ6510 LAN status LEDs



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

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

LAN interface protocols

You can use one of following LAN protocols to communicate with the DAQ6510:

- Telnet
- VXI-11
- Raw socket

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

NOTE

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

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

LAN protocols

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

Raw socket connection

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

VXI-11 connection

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

Telnet connection

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

The DAQ6510 supports the Telnet protocol, which you can use over a TCP/IP connection to send commands to the instrument. You can use a Telnet connection to interact with scripts or send real-time commands.

Dead socket connection

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

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

Reset LAN settings

You can reset the password and the LAN settings with the LXI LAN Reset function.

To reset the DAQ6510 LAN settings:

1. Press the **MENU** key.
2. Under System, select **Communication**.
3. Select the **LAN** tab.
4. Select **LXI LAN Reset**. You are prompted to confirm.
5. Select **Yes**.

A green LAN status LED indicator on the lower left confirms that your instrument was assigned an IP address. Note that it may take several minutes for the computer and instrument to establish a connection.

LAN troubleshooting suggestions

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

- The network cable is in the LAN port on the rear panel of the instrument, not one of the TSP-Link® ports.
- The network cable is in the correct port on the computer. The LAN port of a laptop may be disabled when the laptop is in a docking station.
- The setup procedure used the configuration information for the correct ethernet card.
- The network card of the computer is enabled.
- The IP address of the instrument is compatible with the IP address on the computer.
- The subnet mask address of the instrument is the same as the subnet mask address of the computer.
- Make sure there is no USB cable attached between the instrument and your computer. USB communications take precedence over LAN.

You can also try restarting the computer and the instrument.

To restart the instrument:

1. Turn the power to the instrument off, and then on.
2. Wait at least 60 seconds for the network configuration to be completed.

To set up LAN communications:

1. Press the **MENU** key.
2. Under System, select **Communication**.
3. Select the **LAN** tab.
4. Verify the settings.

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

RS-232

If you have a KTTI-RS232 Communications and Digital I/O Accessory card installed in the instrument, you can communicate with the instrument using an RS-232 interface.

For information on using an RS-232 interface, refer to the documentation for the KTTI-RS232 card.

DAQ6510 web interface

The DAQ6510 web interface allows you to make settings and control your instrument through a web page. The web page includes:

- Instrument status.
- The instrument model, serial number, firmware revision, and the last LXI message.
- An ID button to help you locate the instrument.
- A virtual front panel and command interface that you can use to control the instrument.
- Ability to download data from specific reading buffers into a `.csv` file.
- Administrative options and LXI information.

The instrument web page resides in the firmware of the instrument. Changes you make through the web interface are immediately made in the instrument.

When the LAN and instrument establish a connection, you can open the web page for the instrument.

To access the web interface:

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

NOTE

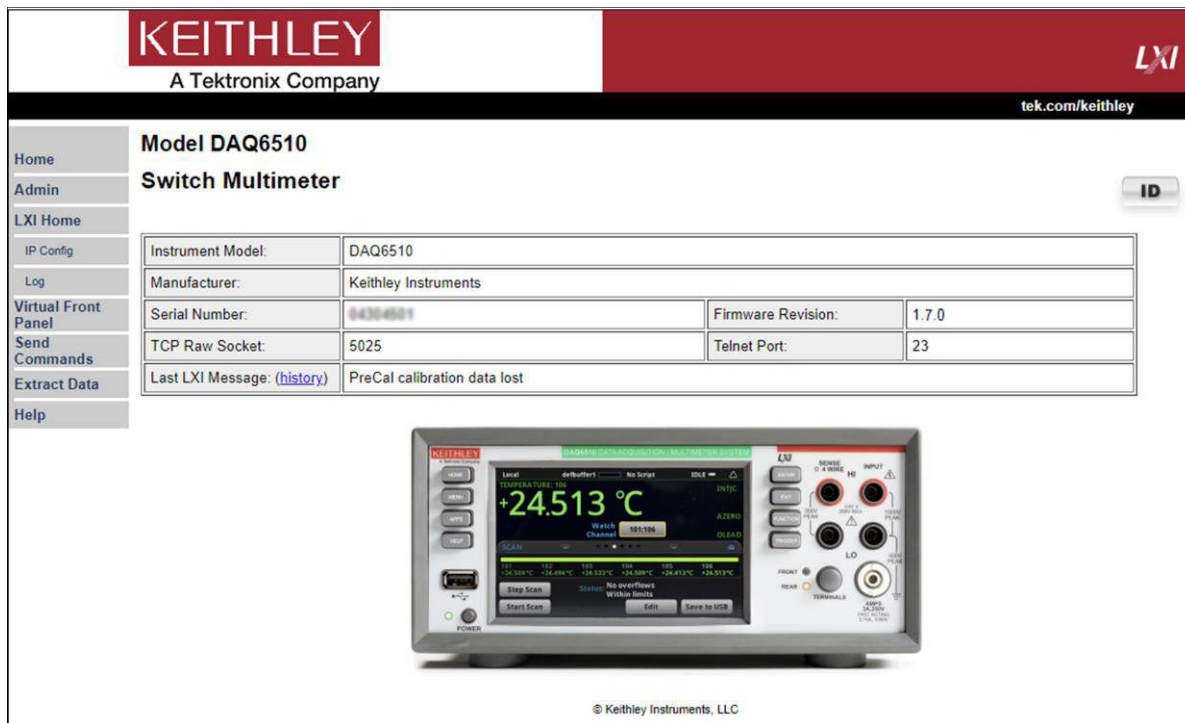
If the web page does not open in the browser, see [LAN troubleshooting suggestions](#) (on page 2-32).

NOTE

To find the IP Address of the instrument, press the Communications indicator in the upper left corner of the home screen.

Web interface Home page

Figure 20: DAQ6510 web interface Home page



The Home page of the instrument provides information about the instrument. It includes:

- The instrument model number, manufacturer, serial number, and firmware revision number.
- The TCP Raw Socket number and Telnet Port number.
- The last LXI message. The history link opens the [LXI Home page](#) (on page 2-35).
- The ID button, which allows you to identify the instrument. Refer to [Identify the instrument](#) (on page 2-35).

Identify the instrument

If you have a bank of instruments, you can select the ID button to determine which one you are communicating with.

To identify the instrument:

1. On the Home page, select the **ID** button. The button turns green and the System Communications menu opens and the LXI LAN indicator on the LAN tab blinks.
2. Select the **ID** button again to return the button to its original color. The LXI LAN indicator stops blinking.

LXI Home page

The LXI Home page displays instrument information, including the host name, MAC address, and VISA resource string. You cannot change the information from this page.

You can use the host name instead of the IP address to connect to the instrument.

It also includes the ID button, which you can use to identify the instrument. See [Identify the instrument](#) (on page 2-35).

Change the IP configuration through the web interface

You can change the LAN settings through the web interface on the instrument. The LAN settings you can change include the IP address, subnet mask, gateway, and DNS address.

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

NOTE

You can also change the IP configuration through the front panel or with TSP and SCPI commands. See [Set up LAN communications on the instrument](#) (on page 2-26) for information.

To change the IP configuration using the instrument web page:

1. Access the internal web page as described in [Connecting to the instrument through the web interface](#) (on page 2-33).
2. From the navigation bar on the left, in the LXI Home menu, select **IP Config**.
3. Select **Modify**. The Modify IP Configuration page is displayed.

Figure 21: Modify IP Configuration web page

Hostname:	K.DAQ6510-000000000
Description:	Kethley DAQ6510 #04304501
TCP/IP Configuration Mode:	☒ Automatic ☐ Manual
Static IP Address:	134.63.78.252
Subnet Mask:	255.255.254.0
Default Gateway:	134.63.78.1
DNS Server:	134.63.75.12
Domain:	global.electronic.net
Submit	

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

NOTE

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

Review events in the event log

Under LXI Home, the Log option opens the event log. The event log records all LXI events that the instrument generates and receives. The log includes the following information:

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

To clear the event log and update the information on the screen, select the **Refresh** button.

Using the DAQ6510 virtual front panel

The Virtual Front Panel page allows you to control the instrument from a computer as if you were using the front panel. You can operate the instrument using a mouse to select options.

The virtual front panel operates the same way as the actual front panel, with the following exceptions:

- The Front/Rear Terminals switch only indicates the setting of the switch. You cannot change which set of terminals is used remotely.
- You cannot switch the instrument on or off with the power switch.
- To scroll up or down on a screen, hold the left mouse button down and swipe up or down.
- To scroll right or left, hold the left mouse button down and swipe left or right. You can also select the dots on the bar above the swipe screens to move from screen to screen.
- You cannot use pinch and zoom on the graph screen.
- You can improve communications speed with the instrument by right-clicking and clearing **High resolution**. The default screen display resolution of 800 × 480 is reduced to 400 × 240 resolution when high resolution is cleared.
- You can display the instrument display only with no other front-panel options by right-clicking and selecting **Screen only**.
- You can download a screen capture by right-clicking and selecting **Download screenshot**.

To use the virtual front panel, you can use any of the standard web browsers. If you are using Microsoft Internet Explorer, it must be version 9 or later. Earlier versions will not allow the swipe motion to work.

NOTE

Using graphing through the virtual front panel requires significant system resources and may slow instrument operation.

NOTE

The DAQ6510 allows fewer than three clients to open the virtual front panel web page at the same time. Only the first successfully connected client can operate the instrument. Other clients can view the virtual front panel.

For information on the options, see [Screen descriptions](#) (on page 3-8).

See the following figure for an example of the virtual front panel.

Figure 22: DAQ6510 virtual front panel



Change the date and time through the web interface

You can change the instrument date and time through the web interface. This is the same as changing the date and time through the front-panel System Settings menu. The date and time are used for the event log entries and data timestamps.

To change the date and time:

1. From the web interface page, select **Admin**.
2. In the Local time table, change the information as needed.
3. Select **Submit**.

Change the password through the web interface

You can change the instrument password from the web interface.

The default user name and password is `admin`. Note that you cannot change the user name; it remains at `admin` even if the password has changed.

To change the password:

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

Send commands using the web interface

You can send individual commands using the web interface.

The active command set is listed above the Command box.

To send commands using the web page:

1. From the navigation bar on the left, select **Send Commands**.
2. If requested, log in.
3. In the **Command** box, enter the command.
4. Select **Send Command** to send the command to the instrument. The command is displayed in the Command Output box. If there is a response to the command, it is displayed after the command.
5. To view any events that have occurred, select **Return Error**.
6. To clear the Command Output list, select **Clear Output**.

Extract buffer data using the web interface

The Extract Data page of the web interface allows you to download reading buffer data from the instrument.

To download buffer data:

1. From the web interface page, select **Extract Data**.
2. In the CSV File column, select the name of the file that you want to download.
3. Follow the instructions for your browser to open the file. Typically, the file opens in Microsoft Excel.

How to install the Keithley I/O Layer

NOTE

Before installing, it is a good practice to check the [Product Support and Downloads web page](https://www.tek.com/support/product-support) (tek.com/support/product-support) to verify that you have the latest version of the Keithley I/O Layer.

You can download the Keithley I/O Layer from tek.com/keithley (tek.com/products/keithley).

To install the Keithley I/O Layer from tek.com/keithley (tek.com/products/keithley):

1. Download the Keithley I/O Layer Software from the Product Support and Downloads web page. The software is a single compressed file.
2. Run the downloaded file from the temporary directory.
3. Follow the instructions on the screen to install the software.
4. Reboot your computer to complete the installation.

Modifying, repairing, or removing Keithley I/O Layer software

The Keithley I/O Layer (KIOL) interconnects many other installers.

To remove all the KIOL components, you need to uninstall the following applications using Control Panel Add/Remove programs:

- NI™ IVI Compliance Package
- NI-VISA Runtime
- IVI Shared Components
- VISA Shared Components
- Keithley SCPI Driver

After uninstalling components, reboot the computer.

Determining the command set you will use

You can change the command set that you use with the DAQ6510. The remote command sets that are available include:

- **SCPI:** An instrument-specific language built on the SCPI standard.
- **TSP:** A scripting programming language that contains instrument-specific control commands that can be executed from a stand-alone instrument. You can use TSP to send individual commands or use it to combine commands into scripts.
- **SCPI2700:** An instrument-specific language that allows you to run code developed for Keithley Model 2700.
- **SCPI2701:** An instrument-specific language that allows you to run code developed for Keithley Model 2701.

If you change the command set, reboot the instrument.

You cannot combine the command sets.

NOTE

As delivered from Keithley, the DAQ6510 is set to work with the SCPI command set.

NOTE

If you choose the SCPI2700 or SCPI2701 command set, you will not have access to some of the extended ranges and other features that are now available using the default SCPI command set. In addition, some Series 2700 code works differently in the DAQ6510 than it did in the earlier instrument. See *DAQ6510 in a Model 2700 or 2701 Application*, Keithley document number 0771465XX, for information about the differences.

To set the command set from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the appropriate **Command Set**.

You are prompted to confirm the change to the command set and reboot the instrument.

To verify which command set is selected from a remote interface, send the command:

```
*LANG?
```

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

```
*LANG SCPI
```

Reboot the instrument.

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

```
*LANG TSP
```

Reboot the instrument.

Interface access

You can specify that the control interfaces request access before taking control of the instrument. There are several modes of access.

You can set one of the following levels of access to the instrument:

- **Full:** Allows full access for all users from all interfaces
- **Exclusive:** Allows access by one remote interface at a time with logins required from other interfaces
- **Protected:** Allows access by one remote interface at a time with passwords required on all interfaces
- **Lockout:** Allows access by one interface (including the front panel) at a time with passwords required on all interfaces

NOTE

The front panel is read-only when you are using a remote interface. You can view information and swipe screens without being prompted to leave remote mode. If you attempt to make a change from the front panel while the instrument is controlled from a remote interface, you will be prompted to enter a password to gain access.

When you set access to full, the instrument accepts commands from any interface with no passwords required. You can change interfaces as needed.

When you set access to exclusive, you must log out of one remote interface before you can log in with another interface. You do not need a password with this access.

Protected access is similar to exclusive access, except that you must enter a password when logging in.

When you set access to locked out, a password is required to change interfaces, including the front-panel interface.

Changing the interface access type

To change the type of interface access from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**. The SYSTEM SETTINGS menu opens.
3. Select **Interface Access**.
4. Select the level of password access control you want to enable.

Using SCPI commands

Send the command that is appropriate for the level of access you want to enable:

```
SYSTem:ACcEss FULL
SYSTem:ACcEss EXCLusive
SYSTem:ACcEss PROTEcted
SYSTem:ACcEss LOCKout
```

Using TSP commands

Send the command that is appropriate for the level of access you want to enable:

```
localnode.access = localnode.ACCESS_FULL
localnode.access = localnode.ACCESS_EXCLUSIVE
localnode.access = localnode.ACCESS_PROTECTED
localnode.access = localnode.ACCESS_LOCKOUT
```

Changing the password

If interface access is set to Protected or Lockout, you must enter a password to change to a new control interface. You can set the password, as described below.

The default password is `admin`.

To change the password from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select **Password**. A keypad opens.
4. Enter the new password.
5. Select the **OK** button on the displayed keyboard. A verification screen is displayed.
6. Enter the new password.
7. Select the **OK** button on the displayed keyboard. The password is reset.

NOTE

You can reset the password by pressing the **MENU** key, selecting **Info/Manage** (under System), and selecting **Password Reset**. When you do this, the password returns to the default setting.

To change the password using SCPI commands:

```
:SYSTem:PASSword:NEW "<password>"
```

Where `<password>` is the new password.

To change the password using TSP commands:

```
localnode.password = "password"
```

Where `password` is the new password.

Switching control interfaces

When the interface access is set to anything other than Full, you need to log in to the instrument from the new interface before you can change any settings. If you have a USB connection and are moving to a LAN connection, you must also disconnect the USB connection. For other communications interfaces, the first interface on which the instrument receives a message takes control of the instrument.

If you are changing to the front panel, when you attempt to make a selection, the Display Lockout - Enter Password keypad is displayed. Enter the password and select the **OK** button on the displayed keyboard.

When you change the remote interface, you must send the following TSP or SCPI command before sending commands:

```
login password
```

Replace *password* with the instrument password.

System information

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

View system information from the front panel

To view the version and serial number information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Info/Manage**.

The firmware version and serial number are displayed.

To view the calibration information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Calibration**.

The instrument displays:

- **Adjust Date:** The date the instrument was adjusted through factory calibration.
- **Adjust Count:** The number of times the instrument has been factory calibrated.
- **Calibration Date:** The date when instrument calibration was last verified.

To view the line frequency information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Scroll down to display the Line Frequency.

View system information using SCPI commands

You can view system information using SCPI commands and common commands.

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

```
*IDN?
```

To read the line frequency, send the command:

```
SYStem:LFRequency?
```

The memory available and factory calibration date are not available when using SCPI commands.

View system information using TSP commands

You can view system information using TSP and common commands.

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

```
*IDN?
```

To read the model number, send the command:

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

To read the serial number, send the command:

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

To read the firmware version, send the command:

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

To read the line frequency, send the command:

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

To view the last verification date of the instrument, send the command:

```
print(cal.verify.date)
```

To view the last adjustment date of the instrument, send the commands:

```
lastCal = cal.adjust.date  
print(lastCal)
```

You can also create user-defined strings to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location. See the [TSP command reference](#) (on page 14-1) for detail about the `userstring` functions.

Instrument description

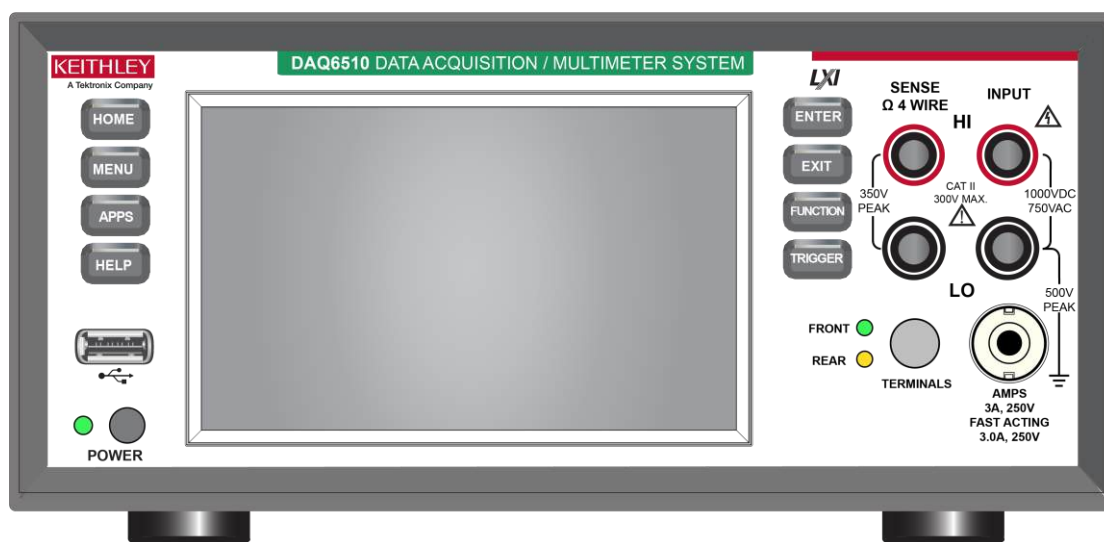
In this section:

Front-panel overview.....	3-1
Rear-panel overview	3-3
Touchscreen display	3-5
Screen descriptions.....	3-8
APPS Manager	3-72
Examples in this manual	3-74
Display features	3-74
Instrument sounds.....	3-80
Using the event log	3-80
Resets.....	3-83

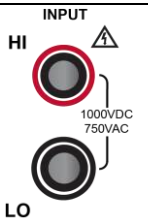
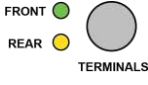
Front-panel overview

The front panel of the DAQ6510 is shown below. Descriptions of the controls on the front panel follow the figure.

Figure 23: DAQ6510 front panel



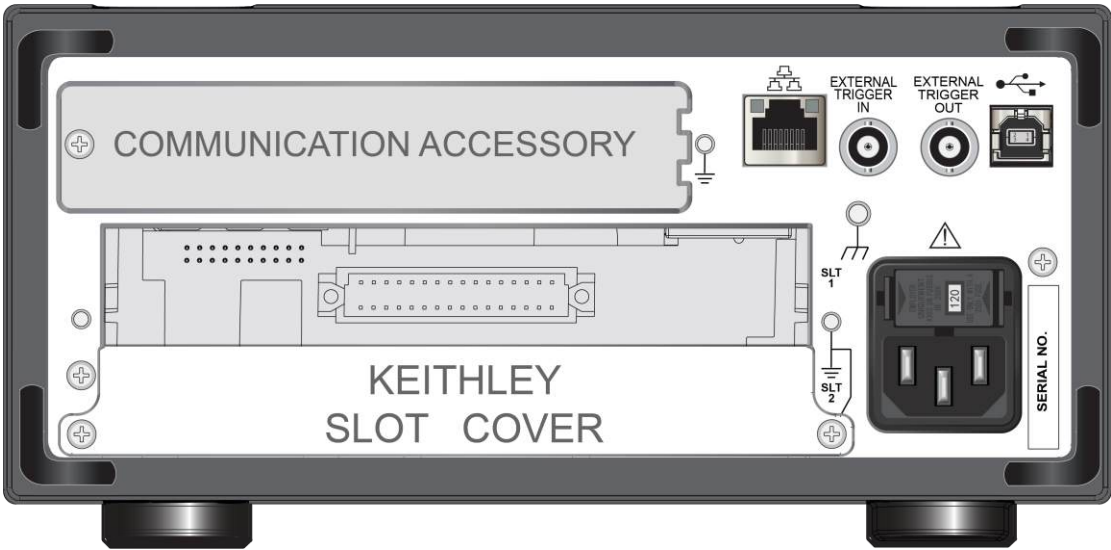
POWER switch		Turns the instrument on or off. To turn the instrument on, press the power switch. To turn it off, press and hold the power switch. The LED is green when the instrument is on and the LED is amber when turned off.
HOME key		Returns the display to the home screen.
MENU key		Opens the main menu. Press the icons on the main menu to open channel, measure, views, trigger, scripts, and system screens. For details, refer to Menu overview (on page 3-29).
APPS key		Opens the APPS Manager. Applications extend the functionality of your DAQ6510.
HELP key		Opens help for the area or item that is selected on the display. If there is no selection when you press the HELP key, it displays overview information for the screen you are viewing. To display help, press and hold the on-screen button while pressing the HELP key.
USB port		Saves reading buffer data and screen snapshots to a USB flash drive. You can also store and retrieve scripts to and from a USB flash drive. The flash drive must be formatted as a FAT or FAT32 drive.
Touchscreen		The DAQ6510 has a high-resolution, five-inch color touchscreen display. The touchscreen accesses swipe screens and menu options. You can access additional screens by pressing the front-panel MENU , APPS , and FUNCTION keys. Refer to Touchscreen display (on page 3-5) for details.
ENTER key		Selects the highlighted choice or allows you to edit the selected field.
EXIT key		Returns to the previous screen or closes a dialog box. For example, press the EXIT key when the main menu is displayed to return to the home screen. When you are viewing a subscreen (for example, the Event Log screen), press the EXIT key to return to the main menu screen.
FUNCTION key		Displays instrument functions. To select a function, touch the function name on the screen.
TRIGGER key		Accesses trigger-related settings and operations. The action of the TRIGGER key depends on the instrument state. For details, see Switching between measurement methods (on page 4-47).
SENSE terminals		Use the SENSE HI and SENSE LO terminals and the INPUT terminals with the 4-wire resistance, 3-wire and 4-wire RTD temperature, and DC voltage ratio functions.

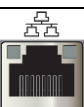
INPUT terminals		Use the INPUT HI and INPUT LO terminals for all measurements except current.
TERMINALS switch		Activates the terminals on the front or rear panel. Selecting the rear panel provides the proper connections to an inserted switching module. When the front-panel terminals are active, the green LED is visible. When the rear-panel terminals are active, the amber LED is visible.
AMPS		Use the AMPS connection with the INPUT LO terminal to measure ≤ 3 A DC or AC_{RMS} current.

Rear-panel overview

The rear panel of the DAQ6510 is shown below. Descriptions of the options follow the figure.

Figure 24: DAQ6510 rear panel



EXTERNAL TRIGGER OUT		This terminal is a TTL-compatible output line with a 0 V to 5 V logic signal. The instrument can generate output trigger pulses on this line. You can use this line for triggering by using the transition of the line state to initiate an action on an instrument monitoring this line. The connector is a BNC type. Refer to External trigger control (on page 8-16) for details.
EXTERNAL TRIGGER IN		This terminal is a TTL-compatible input line with a 0 V to 5 V logic signal. You can trigger the DAQ6510 by using the transition of the line state by another device to initiate an action. The instrument can detect input trigger pulses on this line. The connector is a BNC type. Refer to External trigger control (on page 8-16) for details.
LAN port		Supports full connectivity on a 10 Mbps or 100 Mbps network. The DAQ6510 is a version 1.5 LXI Device Specification 2016 compliant instrument that supports TCP/IP and complies with IEEE Std 802.3 (ethernet LAN). See LAN communications (on page 2-22) for details.
Line fuse and power receptacle		Connect the line cord to the power receptacle and a grounded AC power outlet. The line fuse protects the power line input of the instrument. For safety precautions and other details, see Instrument power (on page 2-5) and Line fuse replacement (on page 10-1).
USB Type B port		USB Type B connection for communications, control, and data transfer. For details, see USB communications (on page 2-16).
Chassis ground		Ground screw for connections to chassis ground. This provides a connection terminal to the equipment frame.
Communications accessory card slot		Installation slot for the communications accessories, which include the Keithley Instruments KTTI-GPIB, KTTI-TSP, and KTTI-RS232.
Keithley Instruments card slots		Two slots are available for the Keithley Instruments Model 77XX series of switching modules.

Touchscreen display

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

CAUTION

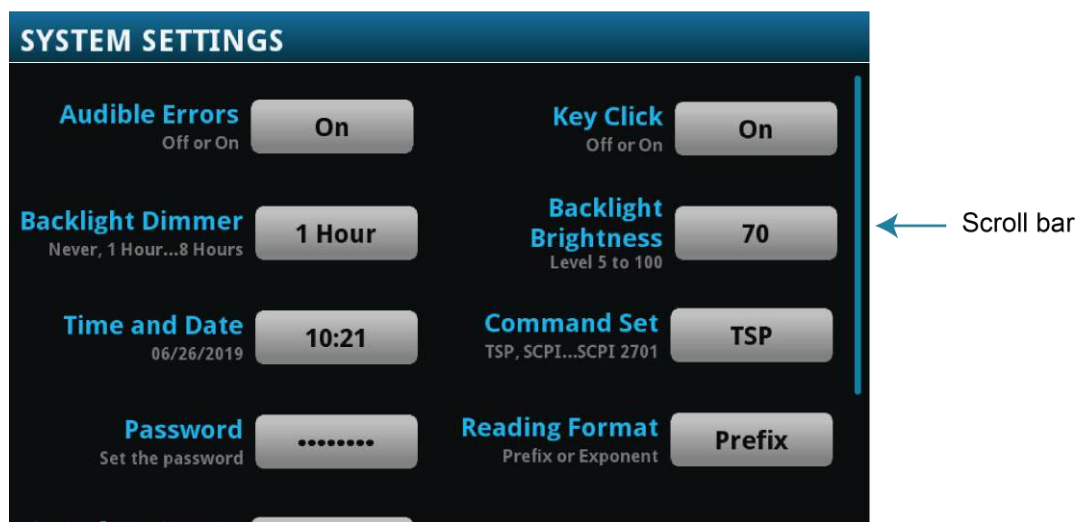
Do not use sharp metal objects, such as tweezers or screwdrivers, or pointed objects, such as pens or pencils, to touch the touchscreen. It is strongly recommended that you use only fingers to operate the instrument. Use of clean-room gloves to operate the touchscreen is supported.

Scroll bars

Some of the interactive screens have additional options that are only visible when you scroll down the screen. A scroll indicator on the right side of the touchscreen identifies these screens. Swipe the screen up or down to view the additional options.

The following figure shows a screen with a scroll bar.

Figure 25: Scroll bar



Enter information

Some of the menu options open a keypad or keyboard that you can use to enter information. For example, if you are setting the name of a buffer from the front panel, you see the keyboard shown in the following figure.

Figure 26: DAQ6510 front-panel keyboard for information entry



You can enter information by touching the screen to select characters and options from the keypad or keyboard. You can move the cursor in the entry box by touching the screen. The cursor is moved to the spot in the entry box where you touched the screen.

Some numeric keypads include Min, Max, and Inf options. Min sets the lowest value for the setting. Max sets the highest value. Inf sets the value to infinite.

Adjust the backlight brightness and dimmer

You can adjust the brightness of the DAQ6510 touchscreen display and buttons from the front panel or over a remote interface. You can also set the backlight to dim after a specified period has passed with no front-panel activity (available from the front-panel display only). The backlight settings set through the front-panel display are saved through a reset or power cycle.

NOTE

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

To adjust the backlight brightness from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select **Backlight Brightness**.
4. Drag the sliding adjustment to set the backlight.
5. Select **OK** to save your setting.

To set the backlight dimmer from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select **Backlight Dimmer**. The Backlight Dimmer dialog box opens.
4. Select a dimmer setting.

To adjust the brightness using the SCPI remote interface, send the following command:

```
:DISPlay:LIgHT:STATe <brightness>
```

Where <brightness> is one of the following options:

- Full brightness: ON100
- 75% brightness: ON75
- 50% brightness: ON50
- 25% brightness: ON25
- Display off: OFF
- Display, key lights, and all indicators off: BLACkout

To adjust the backlight using TSP commands, send the following command:

```
display.lightstate = brightness
```

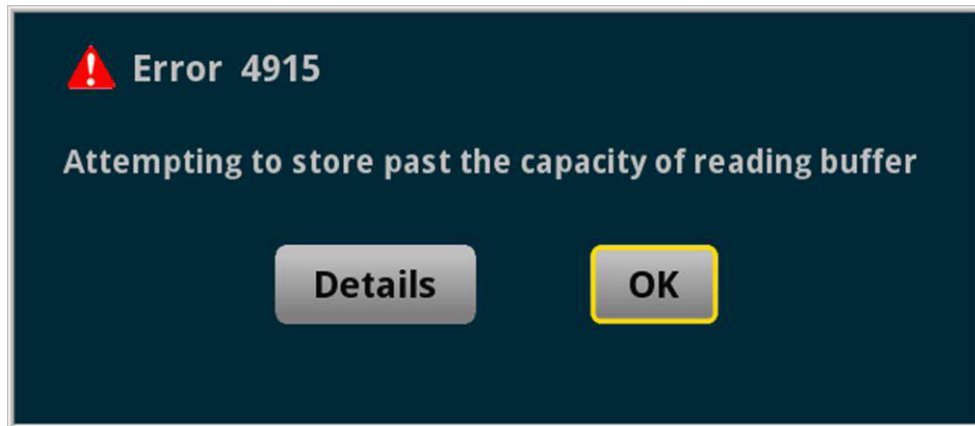
Where *brightness* is one of the following options:

- Full brightness: `display.STATE_LCD_100`
- 75% brightness: `display.STATE_LCD_75`
- 50% brightness: `display.STATE_LCD_50`
- 25% brightness: `display.STATE_LCD_25`
- Display off: `display.STATE_LCD_OFF`
- Display, key lights, and all indicators off: `display.STATE_BLACKOUT`

Event messages

During operation and programming, front-panel messages may be displayed. Messages are information, warning, or error notifications. For information on event messages, refer to [Using the event log](#) (on page 3-80).

Figure 27: Example front-panel error message



Screen descriptions

The following topics describe the screens and options that you can view on the DAQ6510 front-panel display.

NOTE

The information and the available options that appear on the screens will vary depending on whether the TERMINALS switch is set to the front terminals or the rear terminals.

Home screen

This is the default screen that you see whenever you turn the DAQ6510 on or when you press the HOME key.

When the TERMINALS switch is set to FRONT, options for measuring are available.

Figure 28: DAQ6510 home screen - front



When the TERMINALS switch is set to REAR, options for monitoring channels are available in addition to the options for measuring.

Figure 29: DAQ6510 home screen - rear



The options available on the home screen are described in the following topics.

Status and event indicators

The indicators at the top of the home screen contain information about instrument settings and states. Some of the indicators also provide access to instrument settings.

Select an indicator to get more information about the present state of the instrument.

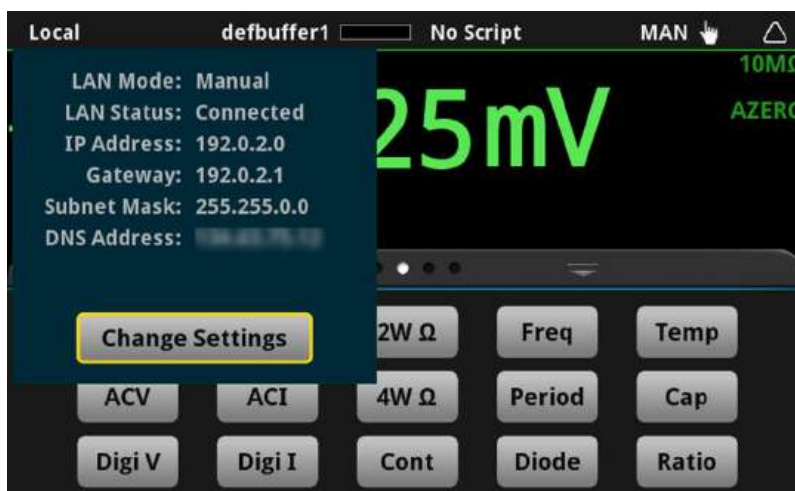
Figure 30: Home screen status bar



Communications indicator

The communications indicator displays the type of communications the instrument is using. Select the indicator to display the present communications settings. Select **Change Settings** at the bottom of the dialog box to open the System Communications screen, where you can change the settings. Refer to [Remote communications interfaces](#) (on page 2-7) for detail on the options that are available.

Figure 31: Communications indicator expanded



NOTE

The options in the following table for RS-232, digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Indicator	Instrument communications
GPIB	Instrument is communicating through a GPIB interface
Local	Instrument is controlled from the front panel
Slave	Instrument is a subordinate in a TSP-Link system
TCPIP	Instrument is communicating through a LAN interface
RS-232	Instrument is communicating through an RS-232 interface
Telnet	Instrument is communicating through Telnet
TSP-Link	Instrument is communicating through TSP-Link
USBTMC	Instrument is communicating through a USB interface
VXI-11	Instrument is communicating through an ethernet interface using the VXI-11 TCP/IP instrument protocol

Communications activity indicator

The activity indicator is located to the right of the communications indicator. When the instrument is communicating with a remote interface, the up and down arrows flash.

Figure 32: Communications indicator



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

Active buffer indicator

The Active Buffer indicator shows the name of the active reading buffer. Select the indicator to open a menu of available buffers. Select a buffer name in the list to make it the active reading buffer. The name of the new active reading buffer is updated in the indicator bar.

The green bar next to the buffer name indicates how full the buffer is.

To create a new buffer, select **Create New**. The new buffer is automatically assigned to be the active buffer.

Figure 33: Active buffer indicator



Active script indicator

This indicator shows script activity and allows you to control script action from the home screen.

If there is no script activity, the indicator displays "No Script."

You can select the indicator to display a menu of available scripts. Select a script name to run that script. You are prompted to confirm that you want to run the script.

If a script is running from the instrument or the USB flash drive, the name of the script is displayed. If a script from TSP is running, `TSP_Script` is displayed. If you select the indicator, you are prompted to abort the running script.

If the instrument is recording a macro script, "Recording" is displayed. You can select the indicator to select an option to stop or cancel recording.

Figure 34: DAQ6510 active script indicator



Measurement method indicator

Located to the right of the active script indicator, this indicator shows the active measurement method. Select the indicator to open a menu. Select one of the buttons on the menu to change the measurement method, initiate or abort the trigger model, or initiate or abort a scan. In the figure below, Continuous Measurement is the present measurement method.

Figure 35: Measurement method indicator



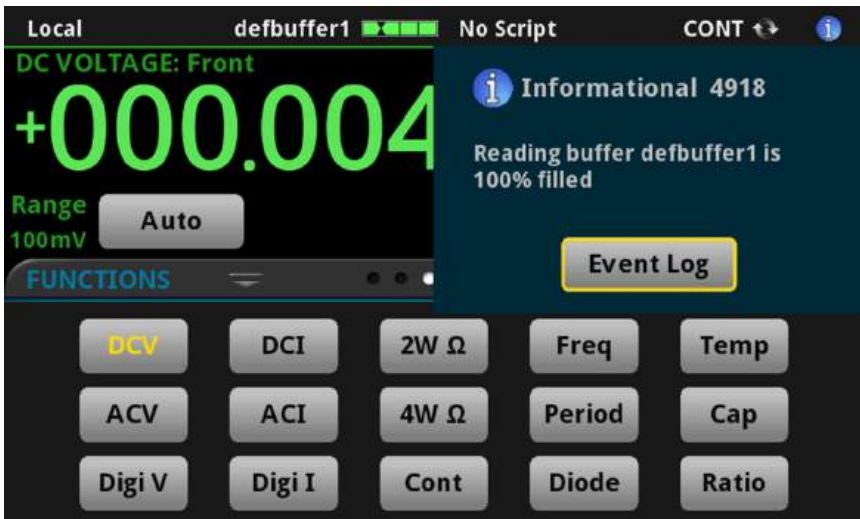
Indicator	Meaning
CONT	Continuous measurement: The instrument is making measurements continuously.
IDLE	Trigger model measurement method. The trigger model is not running.
INACT	The trigger model is inactive. This occurs when the trigger model cannot run, such as when the count is more than the reading buffer capacity or if the buffer is style writable.
MAN	Manual trigger mode: Press the front-panel TRIGGER key to initiate a single measurement.
RUN	Trigger model measurement method. The instrument is running the presently selected trigger model.
WAIT	Trigger model measurement method. The trigger model is waiting on an event.

System event indicator

On the right side of the instrument status indicator bar, this indicator changes based on the type of event that has been logged.

Select the indicator to open a message screen with a brief description of the error, warning, or event. Select the Event Log button to open the System Events tab of the event log, which you can use to access detailed descriptions of the events. For more information about the Event Log, see [Using the event log](#) (on page 3-80).

Figure 36: Error and message indicator



The following table describes the icons.

Icon	Description
	An empty triangle means that no new events were logged in the event log since the last time you viewed the event log.
	A blue circle means that an informational event message was logged. The message is for information only. This indicates status changes or information that may be helpful. If the Log Command option is on, it also includes commands.
	A yellow triangle means that a warning event message was logged. This message indicates that a change occurred that could affect operation.
	A red triangle means that an error event message was logged. When an error occurs, the requested change is not implemented.

Measure view area

The Measure view area of the home screen displays the value of the present measurement and other measurement information.

The options available on the home screen depend on whether you are using the front-panel terminals or rear-panel terminals. When you are using the rear-panel terminals, scan and channel information, including the Watch Channels button, are shown. Watch channels are channels that you want to focus attention on. Watch channels affect what you see on the CHANNEL and STATISTICS swipe screens. They also determine which readings you see on the home screen.

In the Reading Table, you can select the watch channels to filter the buffer so that only information from the watched channels is shown.

In the Graph screens, you can select the watch channels. Each channel in the watch channels list is displayed as a trace on the graph.

Figure 37: Measure view area of the home screen - front terminals selected



Figure 38: Measure view area of the home screen - rear terminals selected



The Range button in the Measure area displays the presently selected measure range. Select the button to change the range.

The indicators on the right edge of the Measure view area show any measure settings that affect the displayed measurement value. The indicators and what they mean are defined in the following table.

Indicator	Meaning
10MΩ	Input impedance is set to 10 MΩ
AUTOΩ	Input impedance is set to automatic
AZERO	Instrument automatically retrieves reference values
DCCPL	DC signal coupling is enabled
EXTJC	The thermocouple reference junction is external
FILT	A filter is applied to the measurement
INTJC	The thermocouple reference junction is internal
L1FAIL	Limit test one is enabled and measurement failed
L1PASS	Limit test one is enabled and measurement passed
L2FAIL	Limit test two is enabled and measurement failed
L2PASS	Limit test two is enabled and measurement passed
MATH	A percent, mx+b, or reciprocal calculation is applied
OCMP	Offset compensation is on
OLEAD	Open lead detection is enabled
REL	Relative offset is applied
SIMJC	The thermocouple reference junction is simulated

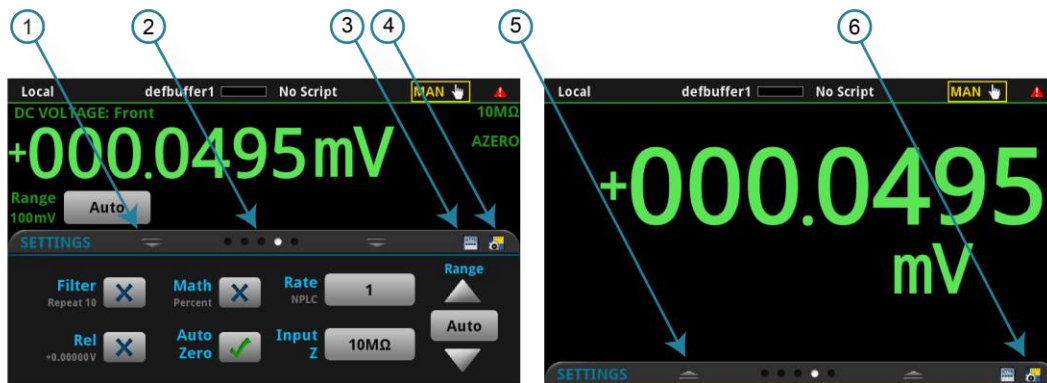
Swipe screens

The DAQ6510 touchscreen display has multiple screens that you can access by swiping left or right on the lower half of the display. The options available in the swipe screens are described in the following topics.

Swipe screen heading bar

The heading bar of the swipe screen contains the following options.

Figure 39: DAQ6510 swipe screens, maximized and minimized



#	Screen element	Description
1	Minimize indicator 	You can swipe down to minimize the swipe screens.
2	Swipe screen indicator 	Each circle represents one swipe screen. As you swipe right or left, a different circle changes color, indicating where you are in the screen sequence. Select a circle to move the swipe screen without swiping.
3	Calculations shortcut 	Select to open the CALCULATION SETTINGS menu. Only available when TERMINALS is set to FRONT.
4	Measure Settings shortcut 	Select to open the MEASURE SETTINGS menu for the selected function. Only available when TERMINALS is set to FRONT.
5	Restore indicator 	Indicates that you can swipe up to display the swipe screen.
6	Graph shortcut 	Select to open the Graph screen.
	Channel Settings shortcut 	Not shown. Select to open the CHANNEL SETTINGS screen. This shortcut is on the settings swipe screen when there is an active closed channel and the terminals are set to rear.
	Scan shortcut 	Not shown. Select to open the SCAN screen. This shortcut is available when the terminals are set to rear.
	Channel control shortcut 	Not shown. Select to open the CHANNEL CONTROL screen. This shortcut is available when the terminals are set to rear.

FUNCTIONS swipe screen

The FUNCTIONS swipe screen highlights the selected measure function and allows you to select a different function.

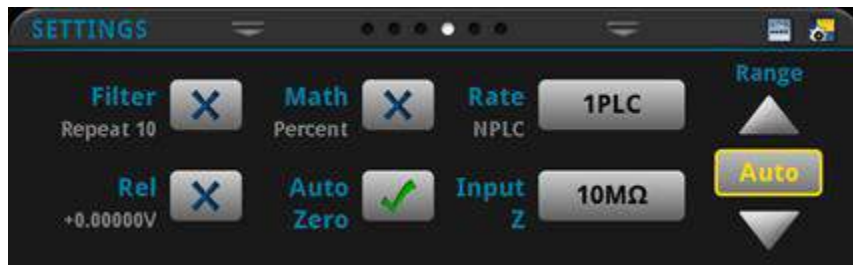
Figure 40: FUNCTIONS swipe screen



SETTINGS swipe screen

The SETTINGS swipe screen gives you front-panel access to some instrument settings. It shows you the present settings and allows you to change, enable, or disable them quickly. The available settings depend on which measure function is active.

Figure 41: SETTINGS swipe screen



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

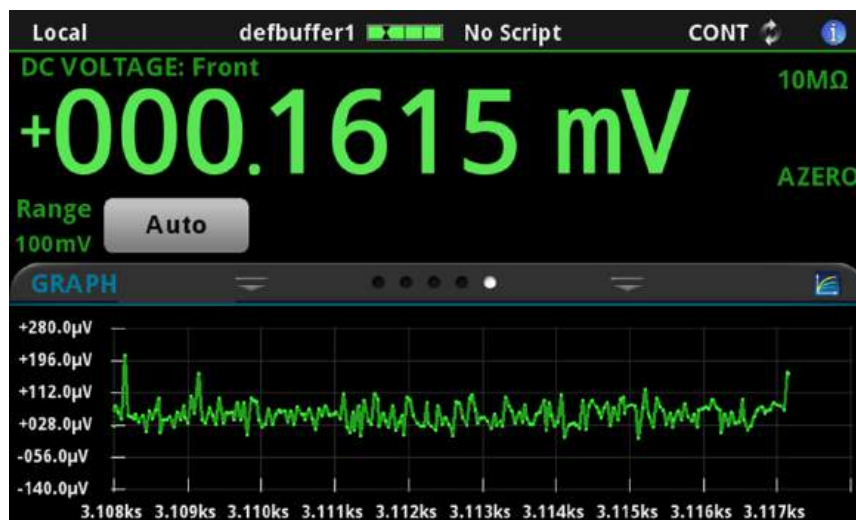
The icons on the right side of the swipe screen heading bar are shortcuts to the CALCULATIONS SETTINGS and MEASURE SETTINGS menus. The CALCULATIONS SETTINGS and MEASURE SETTINGS menus are visible when the FRONT TERMINALS are selected.

For descriptions of the settings, press and hold the on-screen button while pressing the **HELP** key.

GRAPH swipe screen

The GRAPH swipe screen shows a graphical representation of the readings in the presently selected reading buffer.

Figure 42: GRAPH swipe screen



To view the graph on the full screen and to access graph settings, select the graph icon on the right side of the swipe screen header. You can also open the full-function Graph screen by pressing the **MENU** key and selecting **Graph** under Views.

For more information about graphing measurements, see [Graphing](#) (on page 7-1).

STATISTICS swipe screen

The STATISTICS swipe screen contains information about the readings in the active reading buffer. When the reading buffer is configured to fill continuously and overwrite old data with new data, the buffer statistics include the data that was overwritten. To get statistics that do not include data that has been overwritten, define a large buffer size that will accommodate the number of readings you will make. You can use the **Clear Active Buffer** button on this screen to clear the data from the active reading buffer.

If multiple watch channels are set up, you can use the Channel arrows to change the display to show the statistics for each watch channel.

Figure 43: STATISTICS swipe screen



CHANNEL swipe screen

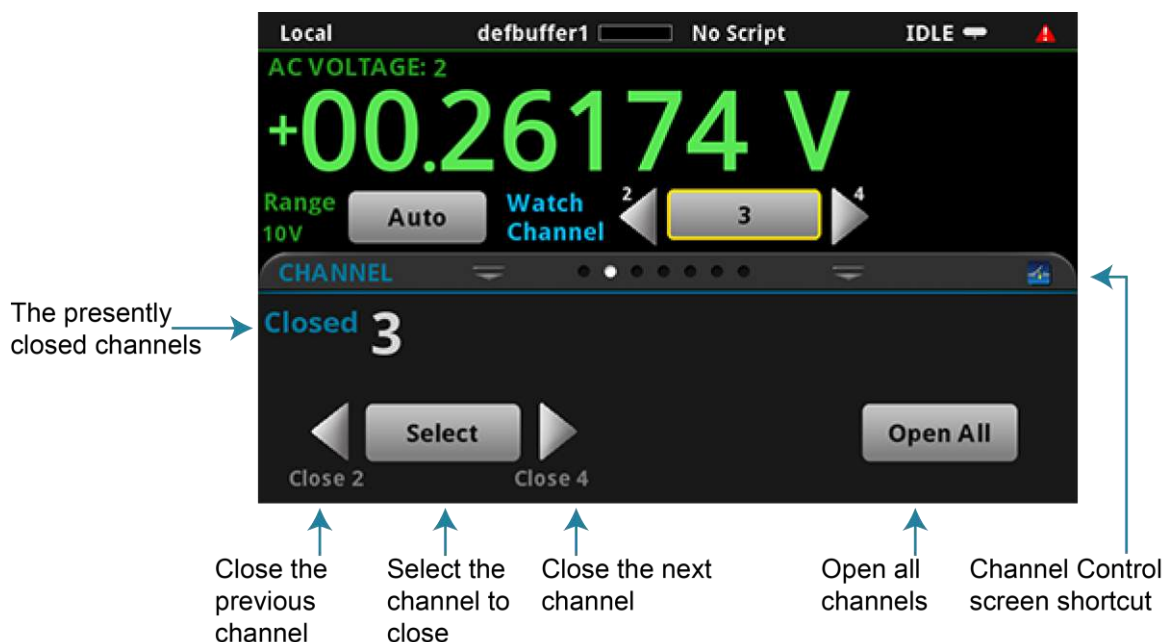
NOTE

The CHANNEL swipe screen is displayed when the TERMINALS switch is set to rear.

The CHANNEL swipe screen gives you front-panel access to channel operation and setup options. The CHANNEL swipe screen provides controls for opening and closing channels on the scanner card. You can select the channel from a list of available channels. You can also open or close all the channels from the channel list.

You can step through the channels on the scanner card. If the channel is a measurement channel and there is no function assigned, you are prompted to assign a function.

Figure 44: Channel swipe screen



NON-SWITCH swipe screen

The NON-SWITCH swipe screen gives you front-panel access to channel Write and Read functions.

The NON-SWITCH swipe screen is only available when the 7706 or 7707 switching module is installed and if the front-panel TERMINALS switch is set to rear. These switching modules have nonswitching channels.

Figure 45: DAQ6510 NON-SWITCH swipe screen



The NON-SWITCH swipe screen has the following control options:

Button	Description
Write	Writes a value to a channel.
Read	Reads a channel.

SCAN swipe screen

The SCAN swipe screen gives you front-panel access to build a scan, edit a scan, start a scan, step through a scan, and display scan results. You can also save the scan results to a USB flash drive.

The icon on the right side of the swipe screen heading bar is a shortcut to the Channel Scan menu. You can also use the Channel Scan menu to build or edit a scan.

When a scan is running, the remaining time and scan count are displayed.

NOTE

This screen is only available when the rear terminals are selected.

Figure 46: SCAN swipe screen



Figure 47: SCAN swipe screen showing scan results



The SCAN swipe screen has the following control options:

Button	Description
Abort Scan	Stop the scan.
Build Scan	Opens the SCAN screen, where you can set up a new scan.
Edit	Opens the SCAN screen, where you can change the setup of a scan.
Pause Scan	Pauses the scan until Resume Scan is selected.
Resume Scan	Resumes a paused scan.
Save to USB	Saves the data in the scan reading buffer to a .csv file on the USB flash drive.
Start Scan	Runs a scan.
Step Scan	Incrementally steps through a scan, channel by channel.

USER swipe screen

You can program custom text and display it on the USER swipe screen. For example, you can program the DAQ6510 to show that a test is in process. This swipe screen is only displayed if custom text has been defined. For details about using remote commands to program the display, refer to [Customizing a message for the USER swipe screen](#) (on page 3-77).

NOTE

This screen is only displayed if custom text is defined using remote commands.

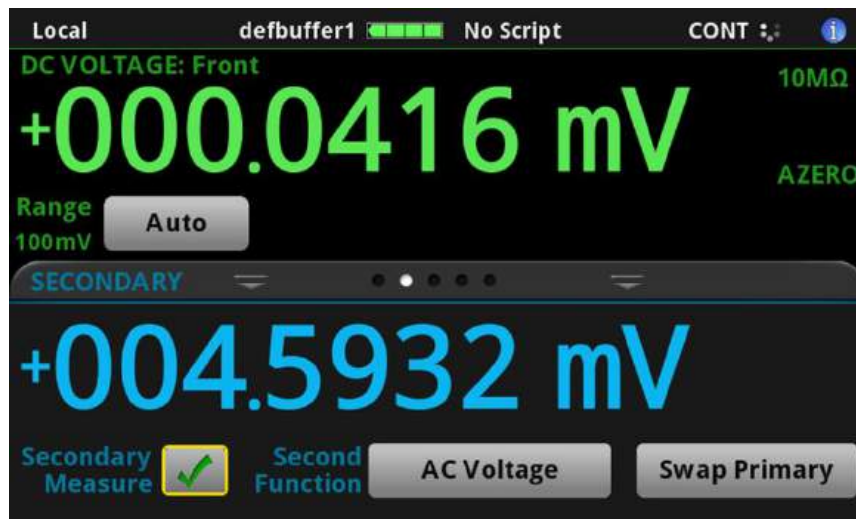
Figure 48: USER swipe screen



SECONDARY swipe screen

The SECONDARY swipe screen allows you to display the results of two measurements on the home screen. Refer to [Display results of two measure functions](#) (on page 4-41).

Figure 49: SECONDARY swipe screen



The secondary measurement window has the following control options:

Button	Description
Secondary Measure	Enables or disables the secondary measurement feature. The primary measurement is not affected by the state of the secondary measurement.
Second Function	Displays the list of functions so you can select the measure function for the secondary measurement.
Swap Primary	Switches the primary and secondary functions.

NOTE

Depending on the selected functions, a relay may click when the instrument switches between the measurement types. Leaving secondary measurements on for extended periods may shorten the life of the relays.

Menu overview

To access the main menu, press the MENU key on the DAQ6510 front panel. The figure below shows the organization of the main menu.

The options on the main menu depend on whether you are using the front terminals or the rear terminals, as shown in the following figures.

Figure 50: DAQ6510 main menu - front terminals selected



Figure 51: DAQ6510 main menu - rear terminals selected



The main menu includes submenus that are labeled in green across the top of the display. Selecting an option in a submenu opens an interactive screen.

Channel menu

The Channel menus allow you to set up and control channels and scans from the front panel.

	The Channel Settings menu contains options to set up the measurement functions for each channel.
	The Channel Control menu contains options to open and close channels.
	The Channel Scan menu contains options to set up and run scans. Options include control of groups, which are channels that are sequential and have the same functions applied to them.

Settings menu



The Channel **Settings** menu contains options to set up the measurement functions for each channel.

Setting	Description
All	Select all channels. Any selections you make in the right pane affect all channels.
Channels	Each channel and its open or closed status is shown. You can select individual or multiple channels. Any selections you make in the right pane affect only the selected channels.
Function	Select the function to assign to the selected channels. When a function is selected, you can set the parameters for that function. You can also select None to have no function assigned to a channel. For detail on the options for each function, refer to Measure Settings menu (on page 3-38). For detail on the calculations that can be applied to the measurements, refer to Measure Calculations menu (on page 3-48).
Slot	Select the slot that contains the channels to which to make changes.
Label	Sets the label associated with a channel. The label must be unique; you cannot assign the same label to more than one channel. Labels cannot start with a digit. They can be up to 19 characters. On the front panel of the instrument, only the first few characters are displayed. After defining a label, you can use it to specify the channel instead of using the channel number in commands.
Channel delay	The channel delay is an additional delay that is added after a channel is closed. You can use this delay to allow additional settling time for a signal on that channel. For most cards, the resolution of the delay is 10 μ s. However, check the documentation for your card to verify. Setting a delay only applies to switch channels.

Control menu



The Channel **Control** menu contains options to open and close channels.

Setting	Description
Channel	Select the switch icon for a channel to close it. Any other closed channels are opened. If no function is selected for the channel, you are prompted to select one.
Open	Select Open to open the selected channel.
Slot	Select the slot that contains the channels you want to control.

Scan menu



The Channel **Scan** menu contains options to set up and run scans. Options include control of groups, which are channels that are sequential and have the same functions applied to them.

Scan options	Description
+ (add group of channels)	Adds a group of channels to the end of the scan list.
 Group menu	The menu icon next to the selected function in the scan list provides the following options: ▪ Insert Channels: Add a channel or group of channels immediately before the selected group. ▪ Change Channel: Allows you to change the channels that are in the group. If you select channels that are assigned to a different function than the other channels in the group, the function is changed to match the group. If you select nonconsecutive channels, a new group with the same function is created. ▪ Delete Group: Removes the selected channel or group from the scan. ▪ Disable Group or Enable Group: Allows you to skip a channel or group during a scan while maintaining the settings. The channel or group is grayed out if it is not available. Select Enable Channel or Enable Group to use the channel or group in the scan again. ▪ Copy Settings: Copies the functions and function settings of a group. ▪ Paste Settings: Pastes copied functions and function settings to another group. If the channels in the group are sequential with another group with the same settings, the groups are merged.
 Scan	The menu icon in the Scan menu bar provides the following options: ▪ Create New List: Removes the existing steps from the scan and allows you to set up a new list. Selections in the Settings, Scan, and Trigger tabs remain. ▪ Reset Scan Settings: Clears the scan list and resets the scan settings to the default values. Measure settings are not affected. ▪ Reset System: Opens the System Info and Management screen, where you can select System Reset. This resets many of the instrument settings to their default values. ▪ Save System: Opens the Save Setup screen, where you can save most of the present settings of the instrument, including scan settings, into a script. ▪ Expand Group: Displays channels in the scan list as individual channels instead of as groups of channels. ▪ Collapse Group: Collapses multiple channels into a single item. Groups are sequential channels with the same functions and settings. Groups allow you to make changes that apply to all channels in the group.
Use Preview	If channels are selected in the Channel Settings menu and have functions set, the selected channels are automatically added to the scan list. To accept this list as the new scan, select Use Preview. You can change the settings after accepting the preview list.

Start	Starts the scan.
Status	Displays the status of the trigger model for the scan: ■ Idle: The scan is stopped. ■ Run: The scan is running. ■ Pause: The scan is paused. ■ Wait: The trigger model of the scan has been in the same wait block for more than 100 ms. ■ Failed: The scan is stopped because of an error. ■ Abort: The scan is stopping because of a user request. ■ Aborted: The scan is stopped because of a user request.

Settings tab

Setting	Description
Function	The function that is set for the selected channel or group of channels.
Measure settings	Settings that are available for the selected function. Refer to Measure Settings menu (on page 3-38) for the available options.
Calculation settings	Settings that specify how measurement information is processed and returned. Refer to Measure Calculations menu (on page 3-48) for the available options.
Label	Assigns a label to the selected channel. The label name is displayed on the home screen and the Channel swipe screen when the channel is displayed. See Assign labels to channels (on page 5-13) for information on setting up labels.
Channel Delay	Sets a channel delay that occurs after the relay closes. This allows extra settling time for the relay. This delay is in addition to normal settling time.

Scan tab

Setting	Description
Scan Count	Sets the number of times the scan is repeated. You can set this to Inf (infinite) which causes the scan to run until the scan is aborted.
Scan to Scan Interval	The interval time between scan starts when the scan count is more than one. If the scan interval is less than the time the scan takes to run, the next scan starts immediately when the first scan finishes.
Scan Duration	The amount of time the scan is expected to take. Scan duration is calculated using the scan-to-scan interval setting and the channel delays.
Buffer	The reading buffer that stores the measurements from this scan. If you select Create New , refer to Measure Reading Buffers menu (on page 3-52) for information on the available options.
Export to USB	Defines when to export the scan data to a USB flash drive. You can select: ▪ After each scan: Export data at the completion of each scan. ▪ Once at end: Export data when all scans are complete. ▪ Never: Do not automatically export scan data. If you select an option that exports data, you are prompted for which data to include, as described below. Time Format: Sets the time format: ▪ Absolute: Each timestamp provides the time and date that the reading was made or the number of seconds from the first buffer reading that the reading was made. ▪ None: No timestamp. ▪ Parts: Timestamps contain dates, hours, minutes, seconds, and fractions of seconds according to Coordinated Universal Time (UTC). ▪ Raw: Timestamps display the absolute time in seconds. ▪ Relative: Timestamps are oriented to a timer with the first buffer reading timestamped at 0.000000 seconds. Each following timestamp is then based on the presently selected format. File Layout: Determines how data is placed in Microsoft® Excel®: ▪ Reading per Row: Readings are displayed in rows. ▪ Reading in Channel Columns: Ignore other columns and use a special format with a column per channel. ▪ Spreadsheet Graph: Ignore other columns and use a special format that is easy to graph in Microsoft Excel. Filename: By default, the file name is the same as the buffer name, followed by the date and time when the file was saved. The date and time are in the format <code>mmdd_hhmmss</code>. To save the file with a different name, select Change. The date and time are not included if you change the file name. Timestamps: Select Each Scan to record a timestamp for the scan. Select Each Reading to record a timestamp for each reading in the scan.

Power Loss Restart	When set to on, causes a scan to automatically restart if it was interrupted by a power failure.
Alarm Limits	Auto Learn runs a scan and establishes alarm limits based on the measurements from the scan. Make sure your system is in a stable state before running Auto Learn. You can also set limits manually. Refer to Setting an alarm limit for scans (on page 5-41) for information on setting alarm limits.
Alarm	When this is on, a trigger is generated when the measurements exceed the limits set for the channels in the scan.

Trigger tab

Setting	Description
Scan Start	Scan Start determines which event starts the scan. You can select: ▪ Monitor Measurement: The scan starts when a measurement above, below, outside, or between specified limit values occurs on a channel. ▪ Digital Input Line: The scan starts when the edge is falling, rising, or either. ▪ TSP-Link Input: The scan starts when a trigger is received from the specified TSP-Link input. ▪ Timer: The scan starts when the specified start time, stimulus, delay, or event count occurs. ▪ Display TRIGGER Key: The scan starts when the front-panel TRIGGER key is pressed. ▪ External Trigger In: The scan starts when a trigger is received on the rear-panel External Trigger In line. ▪ LAN Input Trigger: The scan starts when a LAN In trigger is received. ▪ None: The scan starts immediately. NOTE: Use of a TSP-Link line or a digital output line requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.
Bypass Once	If this option is selected, the scan starts immediately instead of waiting for the channel stimulus event on the first channel of the scan.
Start (Channel)	The channel stimulus event, if any, that must occur before the step action occurs. Starts channel action when a trigger is received. You can select: ▪ Digital Input: Starts channel action when a trigger from a digital input line is received. ▪ TSP-Link Input: The channel action starts when a trigger is received from the specified TSP-Link input. ▪ Timer: Channel action starts when the specified start time, stimulus, delay, or event count occurs. ▪ Display TRIGGER Key: Channel action starts when the TRIGGER key on the front panel of the instrument is pressed. ▪ External Trigger In: Channel action starts when a signal from the EXTERNAL TRIGGER IN line is received. ▪ LAN Input Trigger: Channel action starts when a trigger from a LAN input line is received. ▪ None: Channel action occurs immediately when the scan reaches the channel. NOTE: Use of a TSP-Link line or a digital output line requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Ready (Channel)	Sets a trigger that is sent when the channel is ready. You can select: ▪ Digital Output Line: Sends a trigger through the selected digital output line. ▪ TSP-Link Out Line: Sends a trigger through the selected TSP-Link output line. ▪ Timer: Enables a timer that is triggered when the channel is ready. ▪ External Trigger Out: Sends a trigger to the EXTERNAL TRIGGER OUT line. ▪ LAN Output: Sends a negative or positive trigger to a selected LAN output. ▪ None: No ready trigger is sent. NOTE: Use of a TSP-Link line or a digital output line requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.
Start (Measure)	The measure stimulus event, if any, that must occur before the measure action occurs. Starts measurements when a trigger is received. You can select: ▪ Digital Input: Starts measurements when a trigger from a digital input line is received. ▪ TSP-Link Input: Starts measurements when a trigger is received from the specified TSP-Link input. ▪ Timer: Starts measurements when the specified start time, delay, or event count occurs. ▪ Display TRIGGER Key: Starts measurements when the TRIGGER key on the front panel of the instrument is pressed. ▪ External Trigger In: The measurement starts when a signal from the EXTERNAL TRIGGER IN line is received. ▪ LAN In Trigger: The measurement starts when a trigger from a LAN input line is received. ▪ None: The measurement occurs immediately when the scan reaches the measure/digitize block.
Complete (Measure)	Selects the event that causes a trigger to be asserted when the measurement is complete. You can select: ▪ Digital Out Line: Sends a trigger through the selected digital output line. ▪ TSP-Link Out Line: Sends a trigger through the selected TSP-Link output line. ▪ Timer: Enables a timer that is triggered when the scan starts. ▪ External Out: Sends a trigger to the EXTERNAL TRIGGER OUT line. ▪ LAN Out: Sends a negative or positive trigger to a selected LAN output. ▪ None: No scan complete trigger is sent.
Scan Complete	The event that asserts the trigger to send a notify event when the scan is complete.

Measure menu

The Measure menus allow you to select, configure, and perform measure operations from the front panel. The following topics describe the settings that are available on these screens.

QuickSet menu



The **QuickSet** menu allows you to change the function and adjust performance. This menu is only available if the terminals are set to FRONT.

Setting	Description
Function	Selects the measure function that the instrument uses. Refer to DMM measurement overview (on page 4-4).
Performance	Adjusts the balance between resolution and speed of the instrument. Refer to Using the Performance slider (on page 4-45).

Measure Settings menu



The Measure **Settings** menu contains settings for the presently selected measure function, which is identified by the function indicator in the upper right corner of the menu. The available settings depend on the front-panel **FUNCTION** key selection.

Function indicators

The Function indicator in the upper right corner of some menu screens displays which function the instrument is using to make measurements. The indicators include **DCV Ratio** to indicate that the DC voltage ratio function is selected and **2W Res** to indicate that the 2-wire resistance function is selected. You can select the indicator to open the list of functions and change the active function.

DC voltage measure settings

The following options are available on the Measure Settings menu when the function is set to DC voltage.

Setting	Description
Aperture	Assigns a numerical value to measure the integration rate in seconds; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Auto Zero	Determines if internal reference points are used to maintain stability and accuracy. See Automatic reference measurements (on page 4-53).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Decibel Reference	Sets the decibel reference point; this setting is only available when Unit is set to Decibel. Refer to Show voltage readings in decibels (on page 4-8).
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Input Impedance	Sets impedance to Auto or 10 M Ω . Refer to DC voltage input impedance (on page 4-9).
Integration Unit	Determines if number of power line cycles or aperture is used to set the amount of time the input signal is measured. See Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Line Sync	Enables or disables line synchronization. When it is enabled, it helps increase common-mode and normal-mode noise rejection. Refer to Line cycle synchronization (on page 4-72).
NPLC	Assigns a numerical value for the integration rate to count the number of power line cycles; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).
Unit	Allows voltage to be shown in volts, decibels, or decibel-milliwatts. Refer to Show voltage readings in decibels (on page 4-8) and Show voltage readings in decibel-milliwatts (dBm) (on page 4-8).

AC voltage measure settings

The following options are available on the Measure Settings menu when the function is set to AC voltage.

Setting	Description
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Decibel Reference	Sets the decibel reference point; this setting is only available when Unit is set to Decibel. Refer to Show voltage readings in decibels (on page 4-8).
Detector Bandwidth	Sets the detector bandwidth. Refer to Detector bandwidth (on page 4-52).
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).
Unit	Allows voltage to be shown in volts, decibels, or decibel-milliwatts. Refer to Show voltage readings in decibels (on page 4-8) and Show voltage readings in decibel-milliwatts (dBm) (on page 4-8).

DC current measure settings

The following options are available on the Measure Settings menu when the function is set to DC current.

Setting	Description
Aperture	Assigns a numerical value to measure the integration rate in seconds; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Auto Zero	Determines if internal reference points are used to maintain stability and accuracy. See Automatic reference measurements (on page 4-53).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Integration Unit	Determines if number of power line cycles or aperture is used to set the amount of time the input signal is measured. See Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Line Sync	Enables or disables line synchronization. When it is enabled, it helps increase common-mode and normal-mode noise rejection. Refer to Line cycle synchronization (on page 4-72).
NPLC	Assigns a numerical value for the integration rate to count the number of power line cycles; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).

AC current measure settings

The following options are available on the Measure Settings menu when the function is set to AC current.

Setting	Description
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Detector Bandwidth	Sets the detector bandwidth. Refer to Detector bandwidth (on page 4-52).
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).

2-wire resistance measure settings

The following options are available on the Measure Settings menu when the function is set to 2-wire resistance.

Setting	Description/reference
Aperture	Assigns a numerical value to measure the integration rate in seconds; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Auto Zero	Determines if internal reference points are used to maintain stability and accuracy. See Automatic reference measurements (on page 4-53).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Integration Unit	Determines if number of power line cycles or aperture is used to set the amount of time the input signal is measured. See Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Line Sync	Enables or disables line synchronization. When it is enabled, it helps increase common-mode and normal-mode noise rejection. Refer to Line cycle synchronization (on page 4-72).
NPLC	Assigns a numerical value for the integration rate to count the number of power line cycles; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Offset Compensation	Always set to Off for 2-wire resistance. Refer to Offset-compensated ohms (on page 4-20).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).

4-wire resistance measure settings

The following options are available on the Measure Settings menu when the function is set to 4-wire resistance.

Setting	Description/reference
Aperture	Assigns a numerical value to measure the integration rate in seconds; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73). Available if Integration Unit is set to Aperture.
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Auto Zero	Determines if internal reference points are used to maintain stability and accuracy. See Automatic reference measurements (on page 4-53).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Integration Unit	Determines if number of power line cycles or aperture is used to set the amount of time the input signal is measured. See Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Line Sync	Enables or disables line synchronization. When it is enabled, it helps increase common-mode and normal-mode noise rejection. Refer to Line cycle synchronization (on page 4-72).
NPLC	Assigns a numerical value for the integration rate to count the number of power line cycles; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73). Available if Integration Unit is set to NPLC.
Offset Compensation	Enables or disables offset compensation. Auto is also available. When enabled, offset compensation reduces or eliminates thermoelectric EMFs in low-level resistance measurements. Refer to Offset-compensated ohms (on page 4-20).
Open Lead Detector	Enables or disables open lead detection. When enabled, detects open test leads, which can lead to inaccuracies in 4-wire sensing.
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).

Continuity measure settings

The following options are available on the Measure Settings menu when the function is set to Continuity.

Setting	Description
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Auto Zero	Always set to Off when Continuity is selected.
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Limit 1 High Value	Sets the high value for limit 1. Limit 1 is automatically set to enable. Refer to Limit testing and binning (on page 4-69).
Limit 1 Audible	Determines if the beeper sounds when the resistance is more than or less than the limit 1 high value. Refer to Limit testing and binning (on page 4-69).
Line Sync	Enables or disables line synchronization. When it is enabled, it helps increase common-mode and normal-mode noise rejection. Refer to Line cycle synchronization (on page 4-72).
NPLC	Always set to 0.006 PLC when Continuity is selected.
Range	Always set to 1 k Ω when Continuity is selected.

Frequency measure settings

The following options are available on the Measure Settings menu when the function is set to Frequency.

Setting	Description
Aperture	Controls the amount of time the input signal is measured (aperture), which affects the noise and reading rate; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Range	Always set to Auto when frequency is selected.
Threshold Range	Indicates the expected input level of the voltage signal. You can also set the threshold range to Auto.

Period measure settings

The following options are available on the Measure Settings menu when the function is set to Period.

Setting	Description
Aperture	Controls the amount of time the input signal is measured (aperture), which affects the noise and reading rate; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Range	Always set to Auto when the function is set to Period.
Threshold Range	Indicates the expected input level of the voltage signal. You can also set the threshold range to Auto.

Diode measure settings

The following options are available on the Measure Settings menu when the function is set to Diode.

Setting	Description
Aperture	Controls the amount of time the input signal is measured (aperture), which affects the noise and reading rate; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Auto Zero	Determines if internal reference points are used to maintain stability and accuracy. See Automatic reference measurements (on page 4-53).
Bias Level	Sets the amount of current that is sourced by the instrument to make measurements.
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Integration Unit	Determines if number of power line cycles or aperture is used to set the amount of time the input signal is measured. See Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Limit 1 Audible	Determines if the beeper sounds when the measurement is more than or less than the limit 1 high value. Refer to Limit testing and binning (on page 4-69).
Limit 1 High Value	Sets the high value for limit 1. Refer to Limit testing and binning (on page 4-69).
Limit 1 Low Value	Sets the low value for limit 1. Refer to Limit testing and binning (on page 4-69).
Line Sync	Enables or disables line synchronization. When it is enabled, it helps increase common-mode and normal-mode noise rejection. Refer to Line cycle synchronization (on page 4-72). Always set to Off for diode test.
NPLC	Assigns a numerical value for the integration rate to count the number of power line cycles; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54). Always set to 10 V for diode test.

Temperature measure settings

The following options are available on the Measure Settings menu when the function is set to Temperature.

Setting	Description
2-Wire, 3-Wire or 4-Wire RTD	Sets the RTD type. See Temperature transducer types (on page 4-28).
Aperture	Available if Integration Unit is set to Aperture. Controls the amount of time the input signal is measured (aperture), which affects the noise and reading rate; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Auto Zero	Determines if internal reference points are used to maintain stability and accuracy. See Automatic reference measurements (on page 4-53).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Integration Unit	Determines if number of power line cycles or aperture is used to set the amount of time the input signal is measured. See Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Line Sync	Enables or disables line synchronization. When it is enabled, it helps increase common-mode and normal-mode noise rejection. Refer to Line cycle synchronization (on page 4-72).
NPLC	Available if Integration Unit is set to NPLC. Assigns a numerical value for the integration rate to count the number of power line cycles; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Offset Compensation	RTD only: Enables or disables offset compensation.
Open Lead Detector	Thermocouple and RTDs only: Enables or disables open lead detection. When enabled, detects open test leads.
Reference Junction	Thermocouple only: If rear-panel terminals are selected, you can select Internal, External, or Simulated. Displays <i>Simulated</i> and cannot be changed if front-panel terminals are selected.
RTD Alpha	RTD only when USER is selected: Sets the alpha value of a user-defined RTD.
RTD Beta	RTD only when USER is selected: Sets the beta value of a user-defined RTD.
RTD Delta	RTD only when USER is selected: Sets the delta value of a user-defined RTD.
RTD Zero	RTD only when USER is selected: Sets the zero value of a user-defined RTD.
Temperature	Thermocouple only: The simulated reference temperature.
Thermistor	Thermistor only: Sets the type of thermistor.
Thermocouple	Thermocouple only: Sets the thermocouple type.
Transducer	Sets the type of transducer that is used for temperature measurements. You can use thermocouples, thermistors, 2-wire RTDs, 3-wire RTDs, and 4-wire RTDs with the DAQ6510.
Unit	Sets the type of units that are displayed on the front panel and stored with the temperature measurement in the reading buffer.

Capacitance measure settings

The following options are available on the Measure Settings menu when the function is set to capacitance.

Setting	Description
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).

DC voltage ratio measure settings

The following options are available on the Measure Settings menu when the function is set to DCV Ratio.

Setting	Description
Aperture	Controls the amount of time the input signal is measured (aperture), which affects the noise and reading rate; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Auto Delay	Applies a wait period at the start of a measurement to allow cables and circuitry to settle for best accuracy. Refer to Auto Delay (on page 4-48).
Auto Zero	Determines if internal reference points are used to maintain stability and accuracy. See Automatic reference measurements (on page 4-53).
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Integration Unit	Determines if number of power line cycles or aperture is used to set the amount of time the input signal is measured. See Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Line Sync	Enables or disables line synchronization. When it is enabled, it helps increase common-mode and normal-mode noise rejection. Refer to Line cycle synchronization (on page 4-72).
Method	Remote command only. This command determines if relative offset is applied to the measurements before calculating the DC voltage ratio value. For the SCPI command, refer to [:SENSe1]:<function>:RELative:METHod (on page 12-154). For the TSP command, refer to dmm.measure.rel.method (on page 14-287).
NPLC	Assigns a numerical value for the integration rate to count the number of power line cycles; see Using aperture or NPLCs to adjust speed and accuracy (on page 4-73).
Range	Determines the full-scale input for the measurement in the numerator of the ratio. The range also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).
Sense Range	Displays the full-scale input that is used for the reference measurement in the denominator of the ratio.

Digitize voltage measure settings

The following options are available on the Measure Settings menu when the function is set to digitize voltage.

Setting	Description/reference
Aperture	The measurement aperture time. If the Aperture Type is set to Auto, the aperture time is displayed. If you select Specify, you can set the value.
Aperture Type	Sets the aperture type. You can select Auto or specify a value.
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Decibel Reference	Sets the decibel reference point; this setting is only available when Unit is set to Decibel. Refer to Show voltage readings in decibels (on page 4-8).
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Input Impedance	Sets impedance to Auto or 10 M Ω ; see DC voltage input impedance (on page 4-9).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).
Sample Rate	Sets the number of readings per second.
Signal Coupling	Displays the type of coupling. When DC is displayed, the instrument measures AC and DC components of the signal.
Unit	Sets the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Digitize current measure settings

The following options are available on the Measure Settings menu when the function is set to digitize current.

Setting	Description/reference
Aperture	Sets the measurement data acquisition time window.
Aperture Type	Sets the aperture type to Auto or Specify.
Count	Sets the number of aperture readings that are processed when a measurement is requested.
Display Digits	Sets the number of digits that are displayed for front-panel readings. It does not affect accuracy or speed. Refer to Setting the number of displayed digits (on page 3-75).
Range	Determines the full-scale input for the measurement; also affects the accuracy of the measurements and the maximum signal that can be measured. Refer to Ranges (on page 4-54).
Sample Rate	Sets the number of aperture readings per second.

Measure Calculations menu



The **Calculations** menu contains settings that specify the way measurement information is processed and returned. This menu is only available when the terminals are set to FRONT.

Relative offset

Setting	Description
Rel	Use the relative offset feature to subtract a set value or a baseline reading from measurement readings. When you enable relative offset, all subsequent measurements are displayed as the difference between the actual measured value and the relative offset value.
Rel Value	Sets the relative offset value to be applied to measurements.

Filter

Setting	Description
Count	This sets the number of measurements that are averaged when filtering is enabled.
Filter	Enables or disables the averaging filter for measurements of the selected function. Not available for digitize functions.
Type	Selects the type of averaging filter that is used for the selected measure function when the measurement filter is enabled. Select the moving average filter to continuously add measurements to the stack on a first-in, first-out basis, replacing the oldest measurement in the stack with a new measurement. Select the repeating average filter to average a set of measurements and then flush the data out of the stack before averaging a new set of measurements. Select the hybrid average filter to collect the number of measurements in the count, then begin returning averaged measurements on a first-in, first-out basis. The hybrid filter is only available when the buffer style is set to Full.
Settings	Displays the settings that are available for the averaging filter.
Window	Sets the window for the averaging filter that is used for measurements for the selected function.

Math

Math	This setting enables or disables math operations. When this is on, the math operation specified by Math Format is applied to the measurement.

Setting	Description
b(Offset)	Defines the constant for the offset factor.
m(Scalar)	Defines the constant for the scale factor.
Math Format	When Math is enabled, you can specify which math operation is performed on measurements. You can choose one of the following math operations: ▪ mx+b: Manipulate normal display readings by adjusting the m and b factors. ▪ Percent: Specify a constant that is applied to the measurement and display readings as percentages. ▪ Reciprocal: The reciprocal math operation displays measurement values as reciprocals. The displayed value is $1/x$, where x is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).
Settings	Displays the settings that are available for the math functions.
Zero Reference	When the Math State is set to On, this setting specifies the reference used when the math operation is set to percent: $-1e12$ to $+1e12$.

Limit

Setting	Description
Limit 1 and Limit 2	These settings enable or disable limit testing. The Limit options allow you to do pass-or-fail limit testing using the front panel of the instrument. When you do a limit test, the home screen displays the pass or fail result of the test. Limit State enables or disables a limit test for the selected measurement function. When testing is enabled, limit testing occurs on each measurement. Limit testing compares the measurements to the high and low limit values. If a measurement is outside these limits, the test fails. If a measurement is in the limits, it passes.
Audible	This determines if the instrument beeper sounds when a limit test passes or fails.
Auto Clear	Specifies whether the high and low limits should be cleared automatically.
High Value	The Limit High Level specifies the upper limit for a limit test.
Low Value	The Low Value specifies the lower limit for a limit test.
Settings	Displays the settings that are available for the limit functions.

Measure Config Lists menu



The **Config Lists** menu allows you to select an existing measure configuration list, create a new list, load configuration settings to and from the instrument, and view the settings of an index in a configuration list. The Config Lists menu is only available when TERMINALS is set to FRONT. For more information about using configuration lists, see [Configuration lists](#) (on page 4-99).

Setting	Description
Add Settings	If no index is selected, saves the present instrument settings to an index at the end of the selected configuration list. If an index is selected, you are prompted to append the settings to the end of the list or overwrite the settings at the selected index.
Create New	This option is displayed if no measure configuration lists have been created. When select, it adds a measure configuration list.
Index Details	Displays the details of the selected configuration index. Details include settings such as function, value, delay, calculations, and range. You can scroll the displayed list to view additional settings.
Jump to Index	Opens a number pad that you can use to select an index.
Last Index	Lists the last index in the selected configuration list.
Recall Index	Restores the instrument to the settings stored in the selected configuration list index.
Remove Index	Deletes a configuration list index from the selected configuration list.
Select	Displays a list of available measure configuration lists from which you can choose.

Measure Reading Buffers menu



The **Reading Buffers** menu allows you to view the list of existing reading buffers and select one to be the active buffer. You can also create, save, delete, resize, and clear buffers from this screen.

To create a new reading buffer, select Buffer and select Create New. The new buffer is automatically set to be the active buffer.

Setting	Description
Amount Filled	The percentage of data that is presently in the buffer.
Buffer	Selects an existing buffer to configure. Includes the Create New option, which allows you to create a new buffer.
Capacity	Sets the maximum number of readings that the buffer can store. Note that when you change the capacity of a buffer, the readings in that buffer are cleared.
Clear	Clears data from the selected buffer. You can also press the MENU and EXIT keys simultaneously to clear the active buffer.
Delete	Deletes the selected buffer.
Fill Mode	Continuous: Fills the buffer continuously and overwrites old data when the buffer is full. Once: Stops collecting data when the buffer is full (no data is overwritten).
Make Active	Makes the selected buffer the active reading buffer.
Save to USB	Saves the data in the buffer to a .csv file, which can be opened by a spreadsheet program. A USB flash drive must be present in the front-panel USB port before you select Save to USB.
Style	Defines the amount and type of data the buffer stores. Only available when creating a new buffer. Standard: Store readings with full accuracy with formatting. Full: Store the same information as standard, plus additional information, such as the ratio component of a DCV ratio measurement.

NOTE

The maximum readings represent the highest possible limits and may vary depending on memory usage, reading buffer style, or other reading buffers.

Views menu

The menus under Views in the main menu allow you to select, configure, and view data from measure operations on the DAQ6510. The following topics describe the settings that are available on these screens.

Views Graph menu



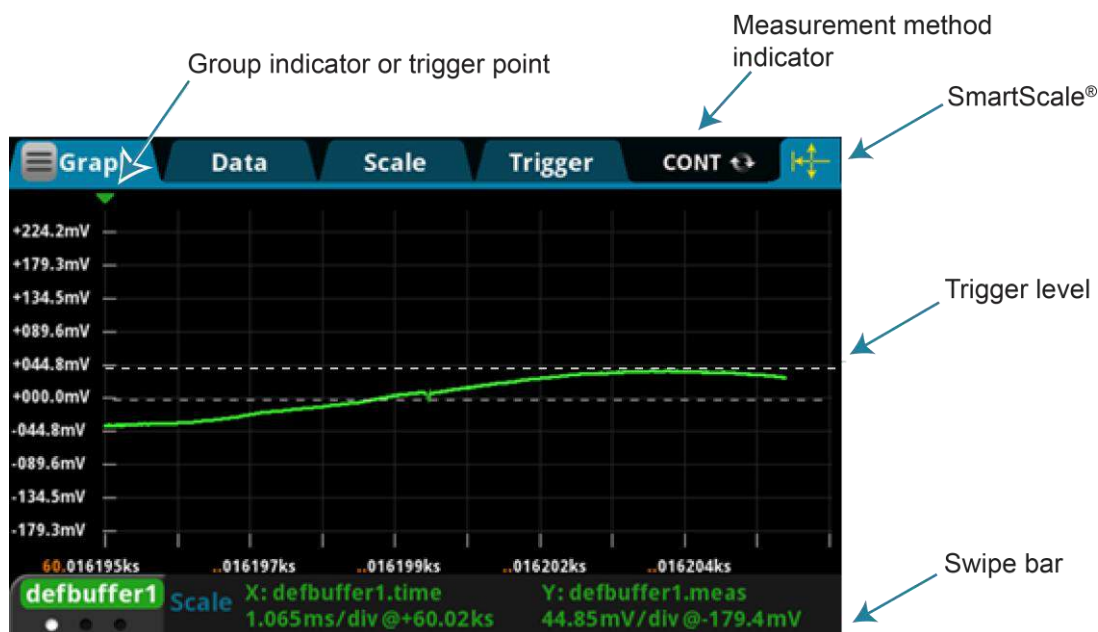
The **Graph** menu opens a screen that displays a graph of the measurements as traces in selected reading buffers. It also contains tabs that you can use to customize the graph display.

You can also select the trigger mode and initiate the trigger model or scan from this screen. Select the measurement method indicator in the upper right corner of the screen and select the measurement method. Refer to [Measurement method indicator](#) (on page 3-14) for details.

Graph tab

The Graph tab graphs readings as they are made by the instrument. Settings you make on the Data, Scale, and Trigger tabs affect how readings appear on this screen. You can also select the number of traces that are displayed.

Figure 52: DAQ6510 graph tab



NOTE

The group indicator does not necessarily mark the location of a trigger event. System latency and programmed delays may cause the first measurement of a group to be displaced in time from its associated trigger event.

NOTE

Trigger levels only apply to DC voltage, DC current, and digitize functions. You can set a trigger level by using an analog trigger waveform as your source event.

You can pinch to zoom in the graph view. You can also swipe the graph to the left or right. When you adjust the view, the SmartScale® feature is turned off. To turn SmartScale back on, select the icon in the upper right of the Graph tab. When SmartScale is on, the instrument determines the best way to scale data based on the data and the instrument configuration (such as the measure count).

The values on the x and y axes show the values set in the Scale tab. The information at the bottom of the Graph tab contains a legend of the active axis and scale settings for the graph. You can swipe the legend to view buffer statistics and readings at the cursors. If the buffer type is set to full and contains extra values, such as when the function is set to DCV Ratio, the statistics are shown as Not Applicable.

Data tab

The Data tab allows you to select the reading buffer that provides the data that is displayed on the Graph tab. You can select up to four buffers. The data from each buffer is shown as a separate trace on the Graph tab. You can also select the type of drawing style that is used on the graph.

Setting	Description
Add	Selects the reading buffers that supply the data for the traces on the Graph tab. You can select up to 20 reading buffers. Colors are automatically assigned to the traces and cannot be changed.
Clear Buffer	Clears data from the selected buffer. To clear the active buffer, you can press the MENU and EXIT keys simultaneously.
Draw Style	The drawing style determines how data is represented when there are many data points. You can select Line, Marker, or Both. When Line is selected, the data points are connected with solid lines. When Marker is selected, the individual data points are shown with no connecting lines. When both are selected, the individual data points are shown and the points are connected with solid lines.
Graph Type	Sets the data to be plotted on the x-axis. You can select Scatter or Time.
Remove	Removes the trace that is selected in the Traces list. This removes the trace from the display.
Traces	Displays the names of the reading buffers that contain the data for the traces that are displayed on the Graph tab. If no buffer is selected, the active buffer is used. You can select up to 20 buffers. The data from each buffer is displayed as a separate trace on the graph. The active buffer contains the data that is displayed on the home screen and where readings are stored when Continuous Measurement is selected or a manual trigger is generated. When an active buffer is selected on the Data tab, that trace tracks the active buffer instead of a specific buffer. If the active buffer changes, the data that is displayed changes to match the new active buffer. To remove the active buffer from the list of traces, select it and select Remove Trace . This does not affect the active buffer that is selected on the home screen. To restore the active buffer to the list of traces, select Add and select the trace labeled Active . When the Terminals switch is set to Rear, Watch Channels is displayed instead of active. Watch Channels allows you to display a group of traces. Groups are selected on the home screen. You can also edit the watch list with the Edit Watch Channels option from the Menu icon on the Graph tab.

Scale tab

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

Setting	Description
Trace	When multiple traces are selected, toggles between the available traces. Information specific to the trace is shown in the same color as the trace.
X-Axis and Y-Axis Minimum Position	Sets the first value that is visible on the graph for the selected trace.
X-Axis and Y-Axis Scale	Sets the reading value scale for each division.
X-Axis Method	The method determines how data is scaled and tracked on the Graph tab. You can select: ▪ SmartScale®: When the SmartScale feature is selected, the instrument scales the graph automatically, determining the best scaling and tracking method based on the data, reading groups, number of traces, and instrument configuration. The scale is set to show the most relevant portion of the data that is in the selected reading buffer. ▪ Show New Readings: The graph always displays the latest data on a fixed scale. ▪ Show Group of Readings: The graph displays a group of readings. A group is automatically created when the measure or digitize count is set to more than 1. ▪ Show All Readings: All data in the buffer is displayed on the graph. ▪ Off: The graph is not automatically adjusted. You can adjust the data manually by swiping, pinching, and zooming. You can also set the Scale and Minimum Position on the Scale tab.
Y-Axis Method	The scale method determines how data is scaled on the Graph tab. If you are graphing one trace, you can select: ▪ SmartScale®: The instrument scales the graph automatically. The scale is set to fit all the data that is in the selected reading buffer onto the screen. The instrument determines the best scale based on the data. ▪ Autoscale Always: Continuously scales the y-axis of the trace so it fits the entire height of the screen. ▪ Autoscale Once: Scales the y-axis of the trace once. ▪ Off: No automatic scaling. If you are graphing multiple traces, you can select: ▪ SmartScale®: The instrument scales the graph automatically. The instrument determines the best scale and tracking method based on the data, reading groups, number of traces, and instrument configuration. ▪ Independent Autoscale: Scales the y-axis of the trace so it fits the entire height of the screen. ▪ Swim Lanes: Scales the y-axis of the traces in equal, nonoverlapping portions of the height of the screen. ▪ Shared Autoscale: Accommodates the minimum and the maximum of all traces. ▪ Off: No automatic scaling.
Y-Axis Scale Format	Sets the scale format that is used on the graph. Select Linear to increase the step size in even increments. Select Log to increase the step size exponentially.

Trigger tab

The Views Graph Trigger tab contains settings that define the trigger mode.

Setting	Description
Source Event	Determines the event that is used to trigger measurements. You can select: ■ Display TRIGGER Key: The trigger occurs when you press the TRIGGER key. External Input Trigger: The trigger occurs when an external pulse is detected. The external pulse can come from a digital input line, TSP-Link input line, or the rear-panel external input line. ■ Waveform: Select an analog edge or window to trigger. ■ None: No trigger event.
Delay	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks.
Position	The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled.
Trigger Clear	This specifies whether previously detected trigger events will be cleared. You can select: ■ Enter: Previously detected trigger events will be cleared. ■ Never: Any previously detected triggers are acted on immediately and not cleared.
Edge	When the source is set to Digital, TSP-Link, or External, this sets the type of edge that generates a trigger. You can set it to Rising, Falling, or Either.
Digital In Line	When the source is set to Digital, this selects the digital input line that will generate the trigger (1 to 6).
TSP-Link Line	When the source is set to TSP-Link, this selects the TSP-Link input line that will generate the trigger (1 to 3).
Level	When the analog edge waveform is selected, sets the signal level that generates the trigger event. The level can be set to any value within the selected measurement range.
Slope	When the analog edge waveform is selected, sets the slope to rising or falling. Rising causes a trigger event when the analog signal trends from below the analog signal level to above the level. Falling causes a trigger event when the signal trends from above to below the level.
Low Boundary	When the analog window waveform is selected, this sets the low level of the analog trigger window.
High Boundary	When the analog window waveform is selected, this sets the high level of the analog trigger window.
Direction	When the analog window waveform is selected, this defines if the analog trigger occurs when the signal enters or leaves the defined high and low analog signal level boundaries.

Views Histogram menu

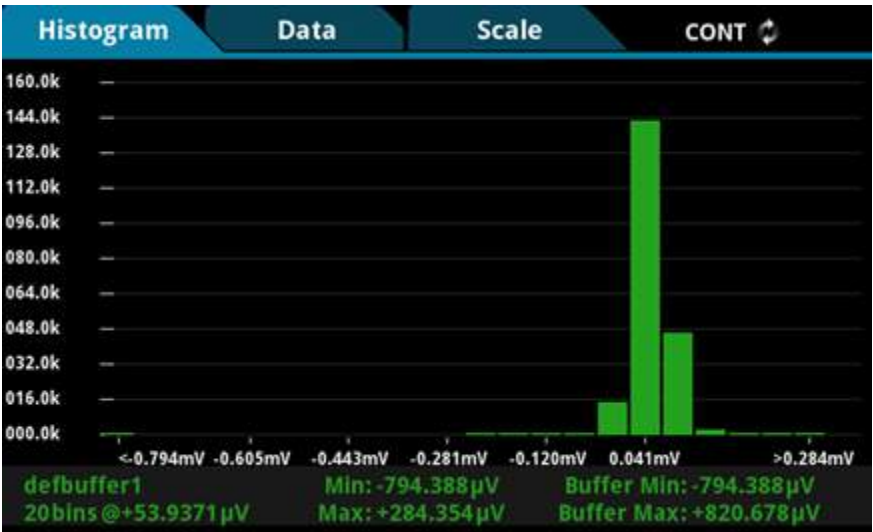


The **Histogram** menu allows you to graph the distribution of measurement data in the selected reading buffer. It also contains tabs that you can use to customize the histogram.

Histogram tab

The Histogram tab graphs readings as a bar graph of the data distribution into bins. Settings you make on the Data and Scale tabs affect which data are used and how data distributions appear on this screen. You can change the scale of either axis on the screen by dragging or pinching the screen.

Figure 53: DAQ6510 Histogram



Data tab

The Data tab allows you to select which reading buffer provides the data that is binned on the Histogram tab. You can also clear the data from the selected buffer.

Setting	Description
Add Trace	Selects the reading buffers that supply the data for the traces on the Data tab. You can specify multiple buffers.
Clear Buffer	Clears data from the selected buffer. To clear data from the active buffer, you can press the MENU and EXIT keys.
Remove Trace	Removes the trace that is selected in the Trace Data list. This removes the trace from the display on the Graph tab.

Scale tab

The Scale tab allows you to set up boundaries, number of bins, and type of scaling used for the histogram.

Setting	Description
Maximum Boundary	The highest value of the data that is binned in the histogram. Data that is above this level is binned in the high outlier bin.
Method	The method of autoscaling to use: ▪ SmartScale®: Automatically select the most appropriate scaling method. ▪ Auto Bin: Redistribute the data evenly in the bins based on the present minimum and maximum boundaries. ▪ Fit: Adjust the y-axis scale so that the tops of all bins are visible ▪ Off: Turn off autoscaling.
Minimum Boundary	The lowest value of the data that is binned in the histogram. Data that is below this level is binned in the low outlier bin.
Number of Bins	The number of bins in the histogram. The histogram will create two outlier bins in addition to the bins you define. These bins are used to collect data that is below or above the defined minimum and maximum boundaries.

Views Reading Table menu



The Reading Table menu allows you to view data in the selected reading buffer.

Setting	Description
Buffer	Selects the reading buffer that contains the data you want to view. If Active is selected, the data from the reading buffer that is presently storing readings is displayed.
Menu	The menu contains options that you can use to view and save data. The following options are only available if TERMINALS is set to REAR: ▪ Filter by Watch Channels (Active Buffer): Filters the data by watch channels. After selecting this option, select Edit Watch Channels to select specific channels. ▪ Edit Watch Channels: Select which channels are watched channels. ▪ Filter by Channels: Allows you to limit the data in the reading table. After selecting Filter by Channels, select Edit Channels to specify the channels to display. ▪ Edit Channels: Allows you to select the channels that are displayed in the reading table. ▪ No Filtering: Removes filters from the reading table and displays all data for the selected reading buffer. The following options are available if TERMINALS is set to either REAR or FRONT: ▪ Jump to Index: Goes to a specific location in the reading table. ▪ Save to USB: Saves the data in the buffer to a .csv file, which can be opened by a spreadsheet program. A USB flash drive must be present in the front-panel USB port before you select Save to USB.
Reading Details	Select a data point to open the Reading Details window for the selected data point. The details describe the instrument settings when the data point was read.
Reading Preview Graph	Shows a small graph view of the data in the reading table. Touch a data point in the graph to jump to that data point in the table.
Table	Displays the data in the selected reading buffer. You can select a data point to display additional detail about that data point.

Trigger menu

The menus under Trigger in the main menu allow you to configure triggering operations from the DAQ6510 front panel. The following topics describe the settings that are available on these screens.

Trigger Templates menu



The **Templates** menu allows you to choose from one of several preprogrammed trigger models. When you select a template, settings you can specify for that template are shown in the lower part of the screen.

You can also customize the templates from the front panel using the Configure menu under Trigger on the main menu screen. For details, see [Trigger Configure menu](#) (on page 3-62).

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

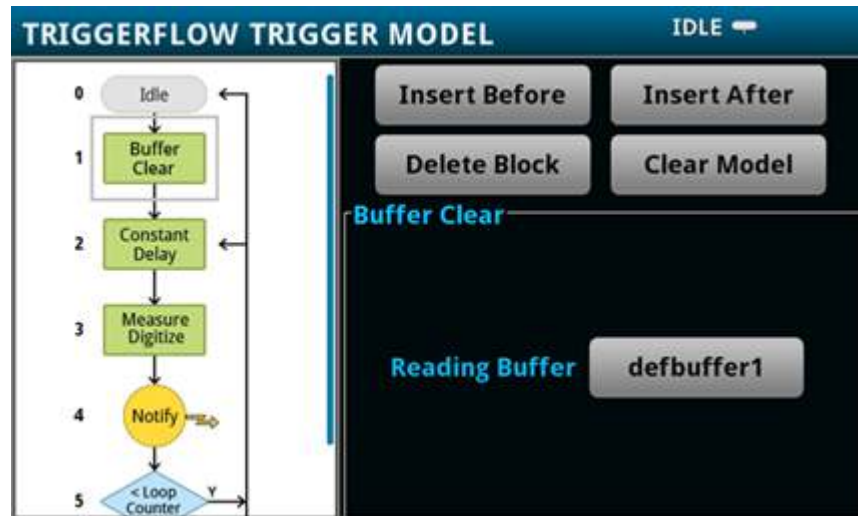
Setting	Description
ConfigList	Creates a trigger model that loads a configuration list. At each configuration list index, a measurement is made. The list is iterated until every index in the configuration list has been loaded.
DurationLoop	Creates a trigger model that makes continuous measurements for a specified amount of time.
Empty	Clears the present trigger model.
GradeBinning	Creates a trigger model that successively measures components and compares their readings to high or low limits to grade components. Only usable if a communications accessory card is installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.
LogicTrigger	Creates a trigger model that waits on an input line, delays, makes a measurement, and sends out a trigger on the output line a specified number of times.
LoopUntilEvent	Creates a trigger model that makes continuous measurements until a specified event occurs.
SimpleLoop	Creates a trigger model that sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you define in the Count parameter.
SortBinning	Creates a trigger model that successively measures components and compares their readings to high or low limits to sort components. Only usable if a communications accessory card is installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger Configure menu



The **Configure** menu allows you to view and modify the structure and parameters of a trigger model. You can also monitor trigger model operation.

Figure 54: TRIGGERFLOW TRIGGER MODEL screen



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

From this screen, you can:

- Insert a new trigger block before or after the selected block
- Choose among several block types to add
- Edit an existing block
- Delete an existing block
- Remove all trigger blocks by selecting Clear Model

When you finish your changes to the trigger model, you can initiate the trigger model by pressing the front-panel TRIGGER key.

For detailed information on the trigger model, refer to [Trigger model](#) (on page 8-35).

Scripts menu

The menus under Scripts in the main menu allow you to configure, run, and manage scripting operations from the DAQ6510 front panel. Scripts are blocks of commands that the instrument can run as a group. The following topics describe the settings that are available on these screens.

Scripts are presented in alphabetic order. Scripts that are on a USB flash drive are presented after scripts that are loaded on the instrument.

Scripts Run menu



The **Run** menu contains a list of scripts that you can select to run immediately. You can also copy a script to a script that runs each time the instrument power is turned on. You can access scripts that are in the instrument or on a USB flash drive.

Setting	Description
Available Scripts	Displays a list of available scripts that you can select. All scripts that are saved on the DAQ6510 or are on a USB flash drive inserted into the instrument are listed.
Copy to Power Up	Saves the selected script to a script that runs automatically when the instrument is turned on. The script is saved with the script name <code>autoexec</code> .
Run Selected	Runs the selected script immediately.

Scripts Manage menu



The **Manage** menu allows you to copy scripts to and from the instrument and the USB flash drive. You can also delete scripts from the instrument or USB flash drive.

Setting	Description
>	Copies a script from the instrument to a USB script. A USB flash drive must be inserted before you select this option.
<	Copies a script from a USB flash drive to the instrument. A USB flash drive must be inserted before you select this option.
Delete	Deletes the script that is selected.

For more information about using scripts with the DAQ6510, see [Fundamentals of scripting for TSP](#) (on page 13-4).

Figure 55: DAQ6510 MANAGE SCRIPTS menu



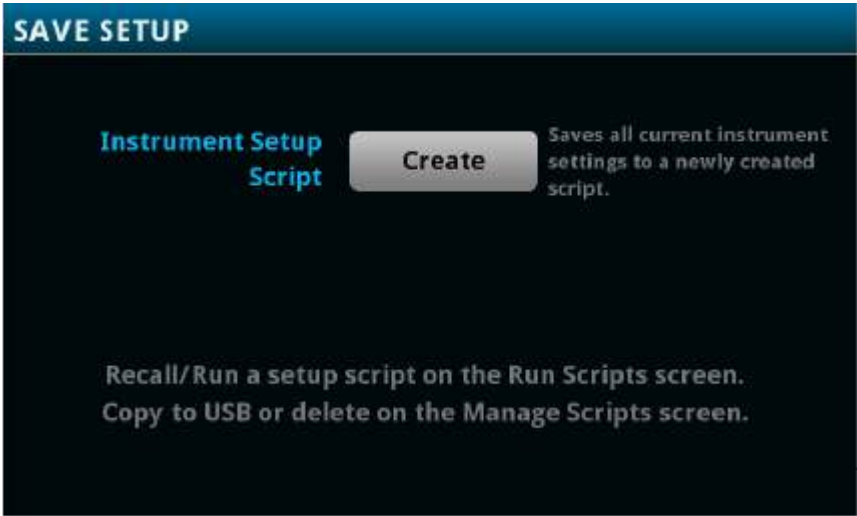
Scripts Save Setup menu



The Save Setup menu allows you to save the present settings and configuration lists of the instrument into a configuration script. You can use this script to recall the settings. Graph settings are not saved.

For more information about user configuration scripts and setups, see [Saving setups](#) (on page 4-92).

Figure 56: DAQ6510 SAVE SETUP menu



Scripts Record menu



The options in the **Record** menu allow you to record your actions and store them in a macro script. The script can be run and managed like any other script using the options in the Scripts menu or remote commands. Note that only settings are stored; no key presses or front-panel only options are stored.

Setting	Description
Cancel Macro	Stop recording without saving.
Start Macro	Begin recording your selections.
Stop Macro	Stop recording. You are prompted to enter a Macro Script Name.

Scripts Apps menu



The Apps menu opens the Apps Manager. Apps Manager is used to manage prebuilt TSP® applications.

TSP applications are Keithley-developed programs that enable the instrument to use specialized functions, test automation, and visualize information on the user interface. TSP applications are available when the instrument is used in the TSP or SCPI command set.

The Local tab shows apps that are installed on the instrument. The USB tab shows apps that are on an installed USB drive.

After selecting an application, select **Run** to run the application. To stop running an application, select **End App** from the top right of the screen.

System menu

The menus under System in the main menu allow you to configure general instrument settings from the DAQ6510 front panel. Among these settings are the event log, communications, calibration, system information, backlight, time, and password settings.

System Event Log menu



The **Event Log** menu allows you to view and clear event log entries. You can also adjust which events are displayed or logged.

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

- **Error:** An error occurred. When an error occurs, the requested change is not implemented.
- **Warning:** This message indicates that a change occurred that could affect operation.
- **Information:** The message is for information only. This indicates status changes or information that may be helpful. If the Log Command option is on, it also includes commands.

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

Setting	Description
Clear Log	Clears all entries from the event log.
Log Command	Turns the logging of commands on or off. When logging is turned on, the instrument records the commands that are sent to the instrument. It records commands sent from any interface (the front panel or a remote interface).
Log Information	Turns the logging of information messages on or off. If this is turned off, the instrument does not log or display popups for information messages.
Log Warning	Turns the logging of warnings on or off. If this is turned off, the instrument does not log or display popups for warning messages.
Popups	Chooses what type and whether to display popup messages on the front panel. You can choose to display error messages, error and warning messages, or no messages in popups. Messages continue to be saved in the event log when popups are turned off.
Reset Popups	Restores the popups setting to show errors and warnings.
Save to USB	Saves the event log to a <code>.csv</code> file on the USB flash drive. The file name is <code>eventlog.csv</code> .
Show Information	Turns the display of information messages on or off. If you turn this off, the instrument continues to record information messages and display popup messages, but does not display them on the System Events tab.
Show Warning	Turns the display of warnings on or off. If you turn this off, the instrument continues to record warnings and display warning popup messages, but does not display them on the System Events tab.

System Communication menu



The **Communication** menu opens a set of tabs that contain information about the communications settings. Most of the tabs contain settings that you can change.

GPIB tab setting	Description
Address	The default GPIB address is 16. You can set the address to any address from 1 to 30 if it is unique in the system. This address cannot conflict with an address that is assigned to another instrument or to the GPIB controller.
GPIB resource string	The VISA instrument connection string is displayed in the lower right.

LAN tab settings	Description
Apply Settings	To save any changes you made on the LAN tab, select Apply Settings .
Gateway	Displays the present gateway address. When TCP/IP Mode is set to Manual, you can set the gateway address.
IP Address	Displays the present IP address. When TCP/IP Mode is set to Manual, you can set the IP address.
LXI LAN Reset	Sets the TCP/IP Mode to Auto and clears the IP address, gateway, and subnet mask. Resets the LAN password.
LXI LAN indicator	The LXI LAN indicator illuminates when the connection is established.
MAC Address	Read-only text that shows the present media access control (MAC) address of the instrument.
Subnet	Displays the present subnet mask address. When TCP/IP Mode is set to Manual, you can set the subnet mask address.
TCP/IP resource strings	The VISA instrument connection strings are displayed in the lower right if the LAN connection is active.
TCP/IP Mode	Select Auto to set the instrument to automatically obtain an IP address. Select Manual to manually set the IP address, gateway, and subnet mask values.

RS-232 tab settings	Description
Baud Rate	Allows you to set the baud rate from 300 to 115200 baud.
Data Bits	The value is fixed at 8.
Flow Control	Displays the flow control settings. You can select None or RTS/CTS. Select RTS/CTS for smoother, more reliable flow control.
Parity	The value is fixed at None.
RS-232 resource string	The VISA instrument connection strings are displayed in the lower right.
Stop Bits	The value is fixed at 1.

TSP-Link tab settings	Description
Node	Sets the TSP-Link node number for the instrument (1 to 63). Each instrument or enclosure attached to the TSP-Link expansion interface is called a node. Each node must be identified with a unique node number. This identification is called a TSP-Link node number.
Initialize	Select Initialize to have the DAQ6510 find all connected TSP-Link instruments and form a network.

USB tab setting	Description
USB resource string	The VISA instrument connection string is displayed in the lower right if the USB connection is active.

System Settings menu



The **Settings** menu contains general instrument settings.

Setting	Description
Audible Errors	Turns the beeper on or off. When the beeper is on, the beeper sounds when an event or error occurs. The audible error setting is not affected by instrument reset or power cycle. For more information, see Instrument sounds (on page 3-80).
Backlight Brightness	Adjusts the brightness of the front-panel display. The sliding adjustment scale adjusts the brightness level.
Backlight Dimmer	Sets the front-panel display to dim after a period (1, 4, or 8 hours) or never.
Command Set	Specifies the type of commands to use when controlling the instrument from a remote interface (SCPI or TSP).
Interface Access	Specifies the request access for control interface before taking control of the instrument: Full, Exclusive, Protected, or Lockout. For details, see Interface access (on page 2-42).
Key Click	Turns the sound that occurs when you press a front-panel key On or Off. The key-click setting is not affected by instrument reset or power cycle.
Line Frequency	Displays the line frequency detected by the instrument. The line frequency is automatically detected and cannot be changed.
Password	Contains the password if the instrument is set to use an access mode that requires a password. The DAQ6510 is programmed with a default user name and password (case-sensitive): - User name: admin - Password: admin You can change the password. See Instrument access (on page 2-42) for more information about controlling access to the instrument.
Reading Format	Sets the format of the front-panel readings to Prefix (adds a prefix to the units symbol, such as k, m, or μ) or Exponent.
Time and Date	Sets the instrument month, day, year, hour, and minute.
Channel Access	Sets the type of access for the backplane and function relays of the DAQ6510. For most applications, this should be set to Simple . For applications where you need to manually control the open and close states of channels, channel pairs, pole relays, multiple multiplexed channels, or the backplane relays, select Full . For more information, see Multiple-channel operation (on page 5-57).

⚠ WARNING

Careless multiple-channel operation could create an electric shock hazard that could result in severe injury or death. Improper operation can also cause damage to the switching modules and external circuitry. Operating channels independently should be restricted to experienced test engineers who recognize the dangers associated with multiple independent channel closures. Do not attempt to perform this procedure unless you are qualified, as described by the types of product users in the [Safety precautions](#) (on page 1-1). Do not perform these procedures unless qualified to do so. Failure to recognize and observe normal safety precautions could result in personal injury or death.

System Calibration menu



The **Calibration** menu displays factory calibration information, including the last adjustment date, the last calibration date, and the number of times the instrument has been adjusted.

Setting	Description
Adjust Count	The adjustment count is the number of times the instrument has been factory calibrated.
Adjust Date	The date when the instrument was adjusted through a factory calibration.
Calibration Date	The date when the instrument calibration was last verified.

System Info/Manage menu



The **Info/Manage** menu gives you access to version and serial number information and settings for instrument firmware and reset functions.

Setting	Description
Downgrade to Older	Returns the DAQ6510 to a previous version of the firmware from a file on a USB flash drive.
Password Reset	Resets the access password to the default value.
License	Displays license agreements.
Serial Number	Displays the serial number of the instrument.
System Reset	Resets many of the instrument settings to their default values.
Upgrade to New	Initiates a firmware upgrade from a file on a USB flash drive.
Version	Displays the version of firmware that is installed in the instrument.

APPS Manager

TSP® applications are Keithley-developed programs that enable the instrument to use specialized functions, test automation, and visualize information on the user interface. TSP applications are available when the instrument is using either the TSP or SCPI command set. Applications may be preinstalled on your instrument.

To access the APPS Manager, press the **APPS** key on the front panel of your DAQ6510.

Selecting either the Local or USB tabs on the APPS Manager screen shows the applications that are installed on the DAQ6510 or on an installed USB drive. After selecting an application from the APPS Manager, select Run to run the application. To stop running an application, select End App from the top right of the screen.

Download and run TSP applications

If an application is removed from your instrument or a new application is made available, you can download the application and install it on your instrument.

To download and run TSP applications from your computer:

1. Download the TSP® application from tek.com/keithley (tek.com/products/keithley).
2. Save and unzip the file onto the root directory of a USB drive.
3. Insert the USB drive into the front panel of the instrument.
4. Press the **APPS** key on the instrument front panel to open the APPS MANAGER screen.
5. Select the **USB** tab in the APPS MANAGER.
6. Select an application. A brief description of the application, including the name, function, and instrument compatibility, is displayed.
7. To run the application, select **Run**.

To save the application to the internal memory as a local application, select the applications and select **Save**. The application is now available on the Local tab.

To delete the application, select the application and select **Delete**.

Examples in this manual

Many of the remote interface examples in this manual show only one function. The features may be available for additional functions.

For example, many allow you to change the display digits.

This SCPI example shows only the DC voltage display digits command:

```
:DISPlay:VOLTage:DIGits 4
```

The example to change the number of displayed digits for TSP is shown as:

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.displaydigits = dmm.DIGITS_4_5
```

You can replace the SCPI `VOLTage` parameter or TSP `dmm.FUNC_DC_VOLTAGE` parameter with the parameter for another function to set the display digits for that function.

The function parameters for SCPI are shown in the following table.

VOLTage[:DC]	TEMPerature	RESistance	VOLTage[:DC]:RATio
VOLTage:AC	CONTinuity	FRESistance	DIGitize:VOLTage
CURRent[:DC]	FREQuency[:VOLTage]	DIODE	DIGitize:CURRent
CURRent:AC	PERiod[:VOLTage]	CAPacitance	

The function parameters for TSP are shown in the following table.

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Display features

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

Setting the number of displayed digits

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

The number of displayed digits does not affect accuracy or speed of the measurements. It also does not affect the format of readings that are returned from a remote command.

From the front panel:

1. Set TERMINALS to **FRONT**.
2. Press **MENU**.
3. Under Measure, select **Settings**.
4. Set **Display Digits**.

This setting takes effect the next time you make a measurement.

From a remote interface:

- SCPI commands: Refer to [:DISPlay:<function>:DIGits](#) (on page 12-50).
- TSP commands: For measure functions, refer to [dmm.measure.displaydigits](#) (on page 14-228). For digitize functions, refer to [dmm.digitize.displaydigits](#) (on page 14-154).

Setting the display format

You can set the format of units that are displayed for measurement readings on the front panel. The formats are:

- **Prefix:** Add a prefix to the units symbol, such as k, m, or μ .
- **Exponent:** Replace the units symbol with exponents.

See the following figures for examples of each display format.

Figure 57: DAQ6510 prefix display format



Figure 58: DAQ6510 exponent display format



From the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the button next to Reading Format.
4. Select the reading format (**Prefix** or **Exponent**).

This setting takes effect the next time you make measurements.

Over a remote interface:

- **SCPI commands:** Refer to [:DISPlay:READIng:FORMat](#) (on page 12-52)
- **TSP commands:** Refer to [display.readingformat](#) (on page 14-133)

Customizing a message for the USER swipe screen

You can customize the message that is displayed on the USER swipe screen.

You must use a remote interface to customize the USER swipe screen.

Creating a message

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

The examples shown here switch the display to the USER swipe screen, set the first line to read `Test in process` and the second line to display `Do not disturb`.

Using SCPI commands:

Send the commands:

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

Using TSP commands:

Send the commands:

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

Clearing the USER swipe screen

You can clear the message that is displayed on the USER swipe screen.

To clear the message using SCPI commands, send the command:

```
:DISPlay:CLEar
```

To clear the message using TSP commands, send the command:

```
display.clear()
```

Creating messages for interactive prompts

If you are using the TSP command language and scripts, you can set up scripts that prompt the operator to enter information from the front-panel display of the instrument.

The options that you can define include:

- Display a number pad so that operator can enter a value.
- Display a custom button that the operator can press.
- Display a message and a predefined set of buttons that the operator can respond to.
- Display a keypad so that the operator can enter information, as shown in the example below.

Figure 59: Input number example



For more information on creating the interactive prompts, see the following command descriptions:

- [display.input.number\(\)](#) (on page 14-123)
- [display.input.option\(\)](#) (on page 14-125)
- [display.input.prompt\(\)](#) (on page 14-127)
- [display.input.string\(\)](#) (on page 14-128)

Save screen captures to a USB flash drive

You can save a screen capture of the front-panel display to a graphic file on a USB flash drive. The instrument saves the graphic file in `.png` file format.

To save a screen capture:

1. Insert a USB flash drive into the USB port on the front panel of the instrument.
2. Navigate to the screen you want to capture.
3. Press the **HOME** and **ENTER** keys. The instrument displays `Saving screen capture`.
4. Release the keys.

Instrument sounds

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

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

To change the beeps when system events occur (setting is only available from the front panel):

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

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

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

To generate an audible tone from the SCPI remote interface, send the command:

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

Where *frequency* is the frequency of the sound in hertz (20 to 20000) and *duration* is the length of the sound in seconds.

To generate an audible tone from the TSP command interface, send the command:

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

Where *duration* is the length of the sound in seconds and *frequency* is the frequency of the sound in hertz (20 to 20000).

Using the event log

The event log records events, which can be errors, warnings, and information reported by the instrument. Through the Event Log menu, you can view these events. You can also specify which events are shown in the event log, which ones are logged, and which ones generate popup messages.

Information provided for each event log entry

Each event log entry includes the following information:

- The date and time when the event occurred in 24-hour time format (MM/DD HH:MM)
- The code number of the event; if you are using a remote interface, you can use this number with the status model to map events to bits in the event registers
- The type of event (separate icons for informational, error, or warning)
- The description of the event

To access an event log listing from the front panel:

1. Press the **MENU** key.
2. Under System, select **Event Log**.
3. Select the **System Events** tab. A list of events is displayed.
4. If the events fill the page, you can scroll down to see additional events.
5. To view additional detail about an event, select the event. A dialog box with additional detail is displayed.

Event log settings

You can set which events you can see in the instrument event log, and which events cause a status message indicator to be displayed on the front panel of the instrument. You can also choose whether or not to log all commands the instrument receives in the event log, which can be useful for troubleshooting problems. You can save the contents of the event log to a USB flash drive. You can clear the event log.

To access event log settings from the front panel:

1. Press the **MENU** key.
2. Under System, select **Event Log**.
3. Select the **Log Settings** tab. A list of settings is displayed.
4. Make the settings as needed.

The options available on this tab are described in the table below.

Settings tab settings	Description
Show Warning	Turns the display of warnings on or off. If you turn this off, the instrument continues to record warnings and display warning popup messages, but does not display them on the System Events tab.
Show Information	Turns the display of information messages on or off. If you turn this off, the instrument continues to record information messages and display popup messages, but does not display them on the System Events tab.
Log Warning	Turns the logging of warnings on or off. If this is turned off, the instrument does not log or display popups for warning messages.
Log Information	Turns the logging of information messages on or off. If this is turned off, the instrument does not log or display popups for information messages.
Log Command	Turns the logging of commands on or off. When logging is turned on, the instrument records the commands that are sent to the instrument. It records commands sent from any interface (the front panel or a remote interface).
Popups	Turns the display of popups on or off. Options are: - Errors: Turn off the display of error popups. - Errors and Warnings: Turn off the display of error and warning popups. - None: Turn off the display of all popups.
Reset Popups	Restores the popups setting to show errors and warnings.
Save to USB	Saves the event log to a <code>.csv</code> file on the USB flash drive. The file name is <code>eventlog.csv</code> .
Clear Log	Clears all entries from the event log.

Effects of errors on scripts

Most errors do not abort a running script. The only time a script is aborted is when a Lua runtime (event code -286, "TSP runtime error") is detected. Runtime events are caused by actions such as trying to index into a variable that is not a table.

Syntax errors (event code -285, "Program syntax") in a script or command prevent execution of the script or command.

Resets

There are several types of resets in the DAQ6510.

In general, the terms "reset," "instrument reset," and "system reset" refer to the reset that is performed when you send the `*RST` or `reset()` command, or when you select **MENU > System > Info/Manage > System Reset** from the front panel. It resets most commands to their default values. Refer to the command descriptions for specifics on which commands are reset by system reset and the default values.

The instrument also responds to other types of resets. These resets include:

- **DMM reset:** This reset is only available if you are using the TSP command set. The `dmm.reset()` function resets any commands that begin with `dmm.` to their default values. Refer to [dmm.reset\(\)](#) (on page 14-327).
- **Password reset:** This resets the instrument password to its default value. You can reset the password by pressing the **MENU** key, selecting **Info/Manage** (under System), and selecting **Password Reset**. When you do this, the password returns to the default setting. Refer to [Instrument access](#) (on page 2-42).
- **Digital line reset:** If you have a KTTI communications card installed in your instrument, this resets digital I/O line values to their factory defaults if you are using the TSP command set. If you are using SCPI, the lines are reset when the system is reset. Refer to [digio.line\[N\].reset\(\)](#) (on page 14-114).
- **LAN reset:** This resets the LAN settings and the instrument password to the system default values. Refer to [Reset LAN settings](#) (on page 2-31).

- **Status preset:** This resets all bits in the status model. If you are using the SCPI command set, refer to [:STATus:PRESet](#) (on page 12-193). If you are using the TSP command set, refer to [status.preset\(\)](#) (on page 14-431).
- **Trigger blender reset:** This reset is only available if you are using the TSP command set. Resets some of the trigger blender settings to their factory defaults. Refer to [trigger.blender\[N\].reset\(\)](#) (on page 14-443).
- **Trigger timer reset:** This reset is only available if you are using the TSP command set. Resets trigger timer settings to their default values. Refer to [trigger.timer\[N\].reset\(\)](#) (on page 14-526).
- **TSP-Link line reset:** This reset is only applicable if you have the KTTI-TSP Communications installed and are using TSP-Link. Resets some of the TSP-Link trigger attributes to their defaults. Refer to [tsplink.line\[N\].reset\(\)](#) (on page 14-545).
- **TSP-Net reset:** This reset is only applicable if you are using TSP-NET. Disconnects all TSP-Net sessions. Refer to [tspnet.reset\(\)](#) (on page 14-560).

Reset the instrument

You can reset many of the instrument settings to their default values. Default values are listed in the command descriptions.

The reading buffer is reset to defbuffer1 when the instrument is reset. Configuration lists are removed when a system reset occurs.

NOTE

If you are connected to a TSP-Link system, resetting the instrument resets all TSP-Link-enabled instruments on the TSP-Link system. If you are using TSP commands and you want to reset only the local instrument, send `localnode.reset()` instead of `reset()`.

NOTE

Reset restores swipe screens to the factory defaults and removes any user-created swipe screens.

Using the front panel:

1. Press **MENU**.
2. Under System, select **Info/Manage**.
3. Select **System Reset**.

The commands are reset and a confirmation message is displayed.

Using SCPI commands, send the command:

```
*RST
```

Using TSP commands, send the command:

```
reset ()
```

Making measurements

In this section:

Test connections	4-1
Measurement overview	4-4
Using Quickset	4-44
Store settings for functions regardless of active state	4-45
Measurement methods	4-46
Auto Delay.....	4-48
Detector bandwidth	4-52
Automatic reference measurements	4-53
Ranges.....	4-54
Relative offset	4-56
Calculations that you can apply to measurements	4-59
Filtering measurement data.....	4-65
Limit testing and binning	4-69
Line cycle synchronization	4-72
Using aperture or NPLCs to adjust speed and accuracy.....	4-73
DMM resistance measurement methods.....	4-75
Low-level voltage measurement considerations.....	4-80
Measurement settling considerations	4-87
Reference junctions	4-88
Order of operations	4-91
Saving setups.....	4-92
Saving front-panel settings into a macro script.....	4-97
Configuration lists.....	4-99

Test connections

WARNING

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

There is no internal connection between protective earth (safety ground) and the LO terminals of the DAQ6510. Therefore, hazardous voltages (more than 30 V_{RMS}) can appear on LO terminals. This can occur when the instrument is operating in any mode. To prevent hazardous voltage from appearing on the LO terminals, connect the LO terminal to protective earth (safety ground) if your application allows it. You can connect the LO terminal to the chassis ground terminal on the front panel or the chassis ground screw terminal on the rear panel. Note that the front-panel terminals are isolated from the rear-panel terminals. Therefore, if you are using the front-panel terminals, ground to the front-panel LO terminal. If using the rear-panel terminals, ground to the rear-panel LO terminal. Failure to follow these guidelines can result in injury, death, or instrument damage.

Be aware that hazardous voltages can appear on the LO terminals even if the terminals are not presently selected. The TERMINALS switch selects the active terminals for the measurement. It does not disconnect the terminals.

The maximum input voltage between INPUT HI and INPUT LO is 1000 V DC and 750 V AC. Exceeding this value may create a shock hazard.

The maximum common-mode voltage (the voltage between INPUT LO and chassis ground) is 500 V_{PEAK}. Exceeding this value may cause a breakdown in insulation that can create a shock hazard.

You can make test connections to the DAQ6510 from the front panel of the instrument.

Some of the switching modules support measurement inputs. If you have a switching module installed, refer to the appropriate switching module documentation, available from tek.com/keithley (tek.com/products/keithley).

Basic connections

You can access the INPUT HI, INPUT LO, SENSE LO, and SENSE HI connections from the front of the instrument. The connections are banana jacks.

The front panel of the instrument shows the maximum allowable voltage differentials between terminals.

The maximum common-mode voltage is the voltage between INPUT LO and ground. You must limit the current from an external common-mode voltage source. You can use protective impedance or a fuse to limit the current.

If you have a switching module installed that supports measurement inputs, reference the appropriate switching module documentation, available from the tek.com/keithley (tek.com/products/keithley).

When making or breaking connections, follow these guidelines:

- Power off the DAQ6510 and all other instruments.
- Disconnect any devices that may deliver energy.
- Make connections to the device under test through a test fixture or other safe enclosure.
- Make sure the DAQ6510 is properly connected to protective earth (safety ground).
- If the test fixture is conductive, make sure the test fixture is properly connected to protective earth (safety ground).
- Make sure the test fixture provides proper protection.
- Properly make interlock connections between the DAQ6510, the test fixture, and any other instruments.
- Make sure to follow all warnings and cautions and to take adequate safety precautions for each set of connections.
- Properly terminate any triaxial cables. All unterminated cable ends must be in a safe enclosure.
- See [Two-wire local sense connections](#) (on page 4-17) and [Four-wire remote sense connections](#) (on page 4-17) for examples of connections.

Front panel or switching module test connections

You can use either the front-panel terminals or the terminals of an optional switching module installed in the rear panel to make connections to the device under test (DUT). The instrument must be set to use either the front or the rear terminals.

If you have a switching module installed that supports measurement inputs, reference the appropriate switching module documentation, available from the tek.com/keithley (tek.com/products/keithley).

NOTE

You cannot make some connections to the front-panel terminals and some to the switching module terminals for the same test setup. All connections for the same test must be made to either the front panel or to the terminals of a switching module.

WARNING

Be aware that hazardous voltages can appear on the LO terminals even if the terminals are not presently selected. The TERMINALS switch selects the active terminals for the measurement. It does not disconnect the terminals.

Setting the instrument to use the front or rear terminals

The selection to use the front or rear terminals must be made using the front-panel switch. There are no remote commands that can be used to set the terminals.

Using the front panel:

Press the **TERMINALS** switch.

When the FRONT LED is lit, the instrument reads from the front-panel terminals. When the REAR LED is lit, the instrument reads from the terminals of a supported switching module installed in the DAQ6510.

Measurement overview

This section describes the connections and basics of making the measurements for each function.

NOTE

The measurement overview presented here assumes that the measurement method is set to Continuous Measurement (the default). Select the Measurement Method indicator to change the measurement method to Continuous, if necessary.

Figure 60: Measurement method indicator



Measurement capabilities

The DAQ6510 can make the following measurements:

- DC voltage measurements from 100 nV to 1000 V
- AC true root-mean-square (RMS) voltage measurements from 0.1 μ V to 750 V
- DC current measurements from 10 pA to 3 A
- AC current measurements from 100 pA to 3 A
- 2-wire resistance measurements from 1 $\mu\Omega$ to 100 M Ω
- 4-wire resistance measurements from 1 $\mu\Omega$ to 100 M Ω
- Continuity measurements from 100 m Ω to 1 k Ω
- Frequency measurements up to a maximum of 300 kHz on voltage signals from 100 mV to 750 V
- Period measurements to a minimum of 333 ms on voltage signals from 5 mV to 750 V
- Diode measurements from 1 μ V to 10 V
- 2-wire, 3-wire, and 4-wire RTD measurements from -200 $^{\circ}$ C to 850 $^{\circ}$ C
- Thermistor measurements from -80 $^{\circ}$ C to 150 $^{\circ}$ C
- Capacitance measurements from 0.1 pF to 100 μ F
- V_{INPUT} measurements from 100 nV to 1000 V; V_{SENSE} measurements from 10 μ V to 10 V
- Digitize voltage measurements from 10 μ V to 1000 V
- Digitize current measurements from 10 nA to 3 A

Warmup time

After the DAQ6510 is turned on, it must be allowed to warm up for at least 30 minutes to allow the internal temperature to stabilize. If the instrument has been exposed to extreme temperatures, allow extra stabilization time.

High-energy circuit safety precautions

CAUTION

To prevent electrical shock and optimize safety when measuring voltage in high-energy distribution circuits, read and use the following information.

Both ends of the cable must be connected before applying any power to the system. Remove all power in the system before connecting the cable to a switching module or external circuitry.

CAUTION

Both D-sub connector shells of this cable must be connected to a safety earth ground. A shock hazard exists when voltage levels are greater than $30 V_{RMS}$, $42.4 V_{PEAK}$, or when 60 V dc is present.

When making measurements in high-energy circuits, use test leads that meet the following requirements:

Test leads should be fully insulated.

Only use test leads that can be connected to the circuit (for example, alligator clips and spade lugs) for hands-off measurements.

Do not use test leads that decrease voltage spacing. These diminish arc protection and create a hazardous condition.

Power circuit test procedure

When testing power circuits:

1. Turn off power to the circuit using the installed connect-disconnect device. For example, remove the device's power cord or turn off the power switch.
2. Attach the test leads to the circuit under test. Use appropriate safety rated test leads for this application. If over 42 V, use double-insulated test leads or add an additional insulation barrier for the operator.
3. Set the DAQ6510 to the proper function and range.
4. Power the circuit using the installed connect-disconnect device and make measurements without disconnecting the multimeter.
5. Remove power from the circuit using the installed connect-disconnect device.
6. Disconnect the test leads from the circuit under test.

DC voltage measurements

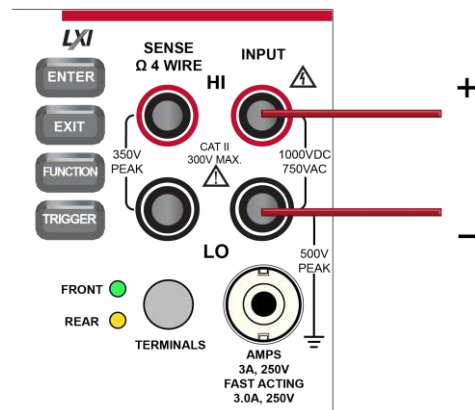
This section describes how you can set up DC voltage measurements.

CAUTION

Inputs: Do not apply more than 1000 VDC between INPUT HI and LO. Failure to observe this caution may result in instrument damage.

DC voltage measure connections

Figure 61: Front-panel connections: DC voltage measurement



Measure DC voltage using the front panel

To make a DC voltage measurement using the front panel:

1. Make the connections as shown in [DC voltage measure connections](#) (on page 4-7).
2. Press the **FUNCTION** key.
3. Select **DC Voltage**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for DC voltage measurements

See [DC voltage measure settings](#) (on page 3-39) for the settings that are available when you are making DC voltage measurements.

Show voltage readings in decibels

You can show DC or AC voltage in decibels (dB), which compresses a large range of measurements into a much smaller scope. The relationship between dB and voltage is defined by the following equation:

$$\text{dB} = 20 \log \left| \frac{V_{\text{in}}}{V_{\text{ref}}} \right|$$

Where:

- V_{in} is the DC or AC input signal
- V_{ref} is the specified voltage reference level

If a relative offset value is in effect when dB is selected, the value is converted to dB, and then relative offset is applied to the dB value. If relative offset is applied after dB has been selected, dB has relative offset applied to it.

NOTE

The largest negative value of dB is -180 dB. This accommodates a ratio of $V_{\text{in}} = 1 \mu\text{V}$ and $V_{\text{ref}} = 1000 \text{ V}$.

Show voltage readings in decibel-milliwatts (dBm)

Decibel-milliwatts (dBm) is used to express an absolute value of power.

To calculate dBm, use the formula:

$$\text{dBm} = 10 \log \frac{(V_{\text{IN}})^2}{R_{\text{REF}} \cdot 1 \text{ mW}}$$

Where:

- V_{IN} is the voltage in
- R_{REF} is the reference impedance

DC voltage input impedance

You can set the input impedance for the DC voltage and digitize voltage functions to automatic (AUTO) or 10 M Ω for all ranges.

Automatic input impedance provides the lowest measure noise with the highest isolation on the device under test (DUT). When automatic input impedance is selected, the 100 mV to 10 V voltage ranges have more than 10 G Ω input impedance. For the 100 V and 1000 V ranges, a 10 M Ω input divider is placed across the HI and LO input terminals.

When the input impedance is set to 10 M Ω , the 100 mV to 1000 V ranges have a 10 M Ω input divider across the HI and LO input terminals. The 10 M Ω impedance provides stable measurements when the terminals are open (approximately 100 μ V at 1 PLC).

Choosing automatic input impedance is a balance between achieving low DC voltage noise on the 100 mV and 1 V ranges and optimizing measurement noise due to charge injection. The DAQ6510 is optimized for low noise and charge injection when the DUT has less than 100 k Ω input resistance. When the DUT input impedance is more than 100 k Ω , selecting an input impedance of 10 M Ω optimizes the measurement for lowest noise on the 100 mV and 1 V ranges. You can achieve short-term low noise and low charge injection on the 100 mV and 1 V ranges with autozero off. For the 10 V to 1000 V ranges, both input impedance settings achieve low charge injection.

When you enable the 10 M Ω input divider, the measurement INPUT HI is connected to INPUT LO. When the input divider is enabled, some external devices (such as high-voltage probes) must be terminated to a 10 M Ω load.

To set input impedance from the front panel:

1. Press the **MENU** key.
2. Select **Measure > Settings**.
3. Select the **Input Impedance** setting.

To set input impedance using SCPI commands:

Refer to [\[:SENSe\[1\]\]:<function>:INPutimpedance](#) (on page 12-141).

Setting input impedance using TSP commands:

For the DC voltage function, refer to [dmm.measure.inputimpedance](#) (on page 14-243).

For the digitize voltage function, refer to [dmm.digitize.inputimpedance](#) (on page 14-157).

AC voltage measurements

This section describes how you can set up AC voltage measurements from the front panel.

CAUTION

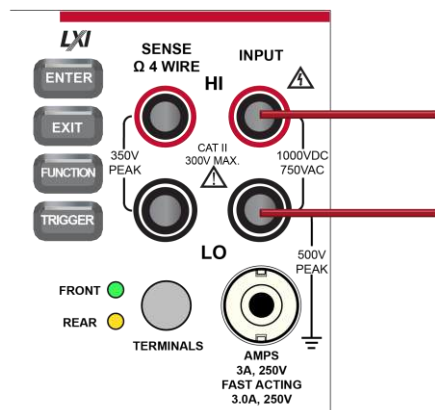
Do not apply more than 750 VAC between INPUT HI and LO. Failure to observe this caution may result in instrument damage.

NOTE

If your application has high voltage, high frequency input signals, you may also need to shield the source to prevent problems with the DAQ6510 display. Refer to [Shielding](#) (on page 4-85) for more information.

AC voltage measure connections

Figure 62: Front-panel connections: AC voltage measurement



Measure AC voltage using the front panel

To make an AC voltage measurement using the front panel:

1. Make the connections as shown in [AC voltage measure connections](#) (on page 4-10).
2. Press the **FUNCTION** key.
3. Select **AC Voltage**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for AC voltage measurements

See [AC voltage measure settings](#) (on page 3-40) for settings that are available when you are making AC voltage measurements.

Show voltage readings in decibels

You can show DC or AC voltage in decibels (dB), which compresses a large range of measurements into a much smaller scope. The relationship between dB and voltage is defined by the following equation:

$$\text{dB} = 20 \log \left| \frac{V_{\text{in}}}{V_{\text{ref}}} \right|$$

Where:

- V_{in} is the DC or AC input signal
- V_{ref} is the specified voltage reference level

If a relative offset value is in effect when dB is selected, the value is converted to dB, and then relative offset is applied to the dB value. If relative offset is applied after dB has been selected, dB has relative offset applied to it.

NOTE

The largest negative value of dB is -180 dB. This accommodates a ratio of $V_{\text{in}} = 1 \mu\text{V}$ and $V_{\text{ref}} = 1000 \text{ V}$.

Show voltage readings in decibel-milliwatts (dBm)

Decibel-milliwatts (dBm) is used to express an absolute value of power.

To calculate dBm, use the formula:

$$\text{dBm} = 10 \log \frac{\frac{(V_{\text{IN}})^2}{R_{\text{REF}}}}{1 \text{ mW}}$$

Where:

- V_{IN} is the voltage in
- R_{REF} is the reference impedance

DC current measurements

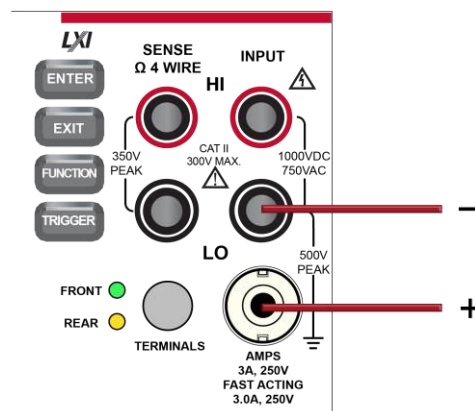
This section describes how you can set up DC current measurements from the front panel.

⚠ WARNING

To prevent electric shock, never make or break connections while power is present in the test circuit.

DC current measure connections

Figure 63: Front-panel connections: DC current measurement (3 A or less)



Measure DC current using the front panel

To make a DC current measurement using the front panel:

1. Make the connections as shown.
2. Press the **FUNCTION** key.
3. Select **DC Current**.
4. Press the TRIGGER key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for DC current measurements

See [DC current measure settings](#) (on page 3-40) for settings that are available when you are making DC current measurements.

AC current measurements

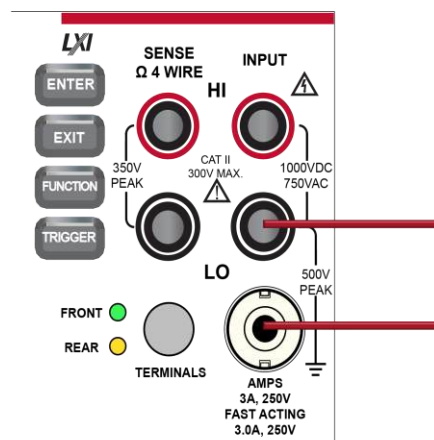
This section describes how you can set up AC current measurements from the front panel.

⚠ WARNING

To prevent electric shock, never make or break connections while power is present in the test circuit.

AC current measure connections

Figure 64: Front-panel connections: AC current measurement (3 A or less)



Measure AC current using the front panel

To make an AC current measurement using the front panel:

1. Make the connections as shown.
2. Press the **FUNCTION** key.
3. Select **AC Current**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for AC current measurements

See [AC current measure settings](#) (on page 3-41) for settings that are available when you are making AC current measurements.

Resistance measurements

You can make 2-wire or 4-wire resistance measurements with the DAQ6510.

For resistances more than 10 k Ω , the two-wire method is typically used for measurements. For resistances less than 10 k Ω , use the 4-wire measurement method to cancel the effect of test-lead resistance.

CAUTION

Do not apply more than 1000 VDC between INPUT HI and LO. Failure to observe this caution may result in instrument damage.

For high-resistance measurements in a high-humidity environment, use Teflon™ insulated cables to minimize errors due to cable leakage.

Two-wire compared to four-wire measurements

You can use 2-wire or 4-wire measurement techniques with the DAQ6510.

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

- Low-impedance applications
- When measuring resistance that is less than 10 k Ω

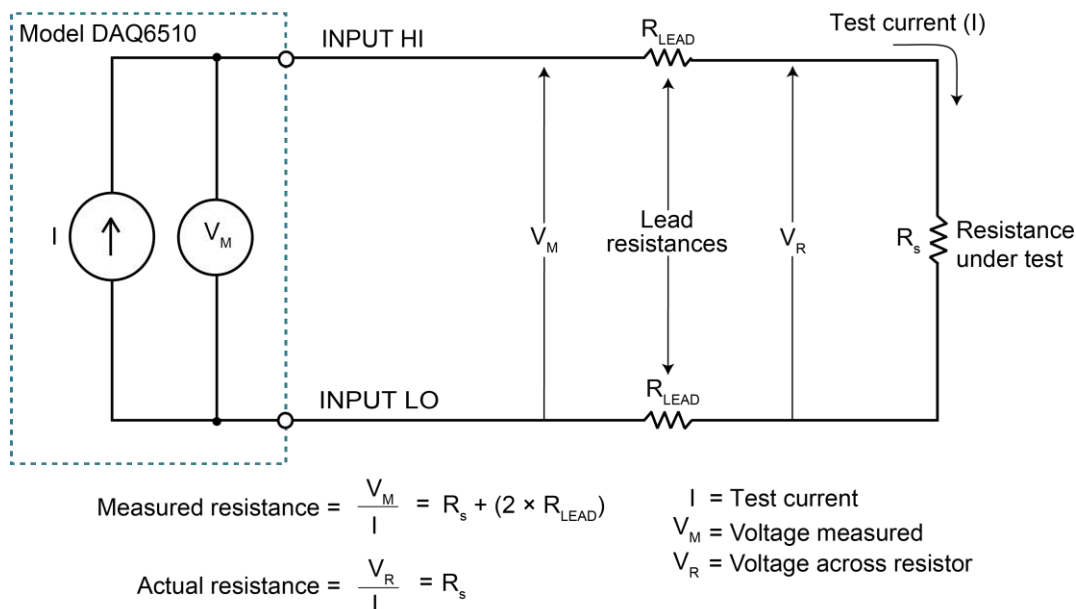
Use 4-wire connections when you are concerned about voltage drops because of lead or contact resistance that could affect measurement accuracy. This can occur on low-impedance devices when you are measuring through a relay switch card.

You can use 2-wire, or local sensing, measurement techniques when the voltage drop due to the 2-wire test current and cable lead resistance is minimal compared to the resistance of the device under test.

Accuracy of 2-wire resistance measurements

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

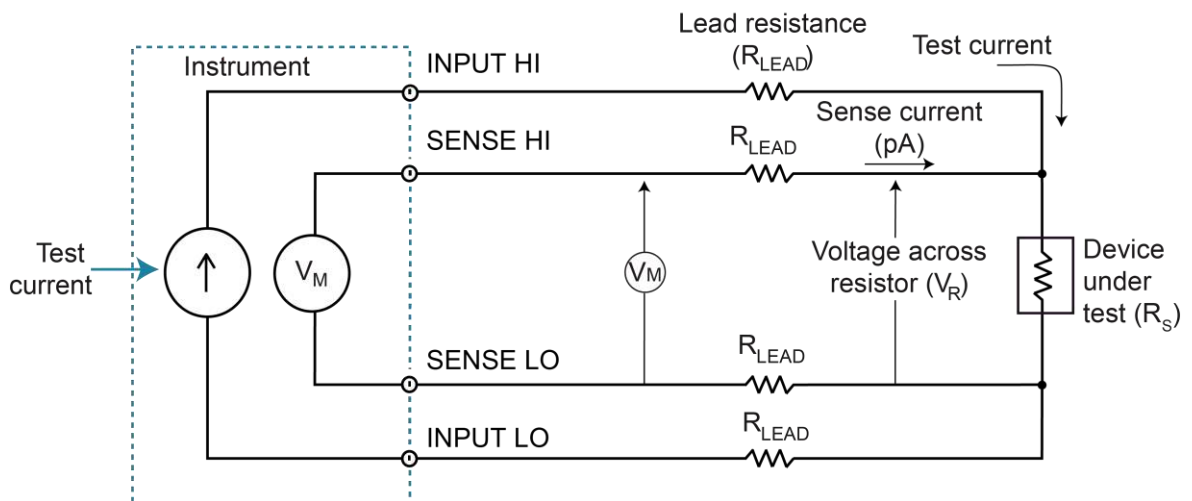
Figure 65: Two-wire resistance sensing for a high-impedance DUT



Minimizing the effect of lead resistance with 4-wire testing

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

Figure 66: DAQ6510 4-wire resistance sensing



Sense current is negligible, therefore $V_M = V_R$

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

Open lead detection

When 4-wire measurements are made, erratic readings can occur if the Sense HI, Sense LO, or both terminals are open. This can be caused by broken test leads.

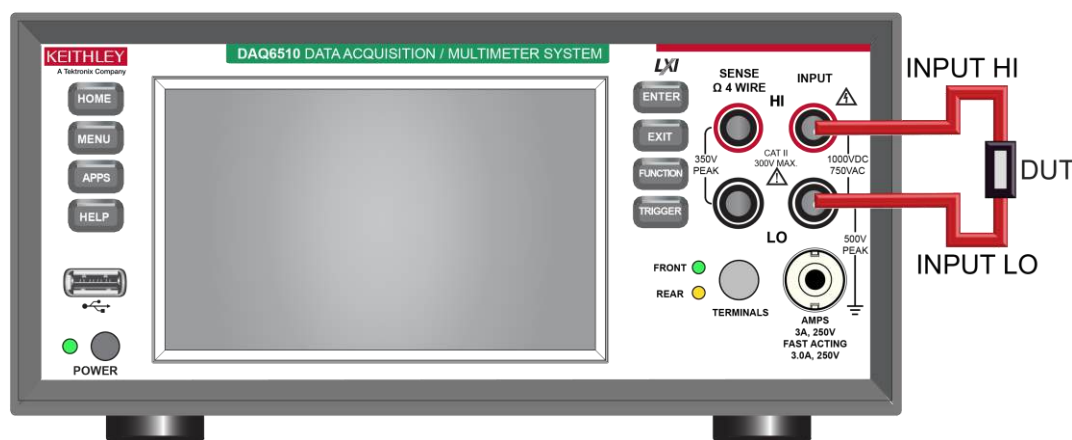
To prevent erratic readings from open leads, you can enable the open lead detector feature. When open lead detection is enabled and the range is 1 Ω to 1 M Ω ranges, the instrument pulses a 1 ms negative current on the Sense HI and Sense LO terminals. If the signal at either terminal is less than -10 mV, the display reads `Overflow $\Omega$` . If the signal is more than -10 mV, the current pulse is automatically shut off, and the 4-wire measurement continues. For the 10 M Ω to 100 M Ω ranges, only the Sense LO terminal is pulsed with a negative current, which minimizes settling time and device-under-test noise.

When open lead detection is enabled, there is minimal impact on reading rates and an increase in measurement reliability and integrity. Open lead detection reduces the reading rate by 2 ms while Sense HI and Sense LO are measured. For measurements made through long capacitive cables or switch cards, the open lead detection pulse current can increase settling time and decrease accuracies, especially for the 10 k Ω to 1 M Ω ranges.

Two-wire local sense connections

Two-wire connections are shown in the following figure.

Figure 67: Two-wire DUT connections to the front panel



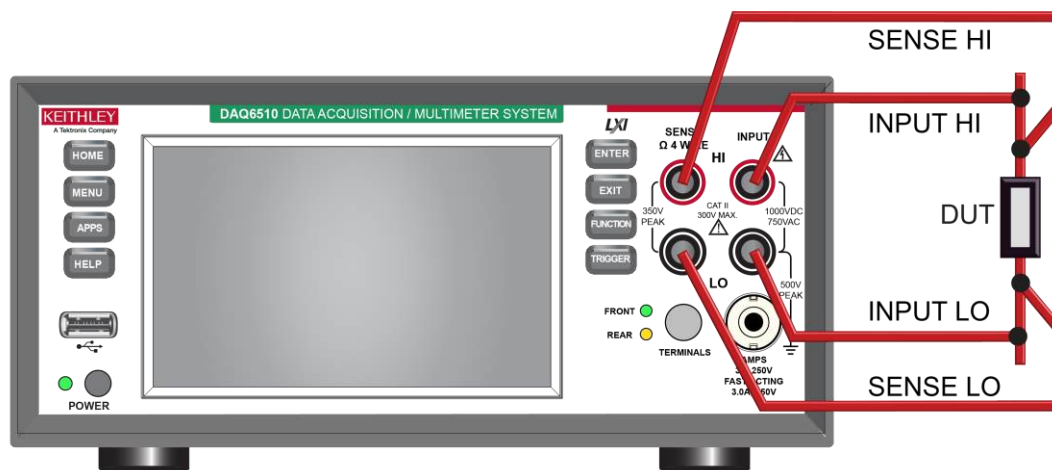
Four-wire remote sense connections

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

NOTE

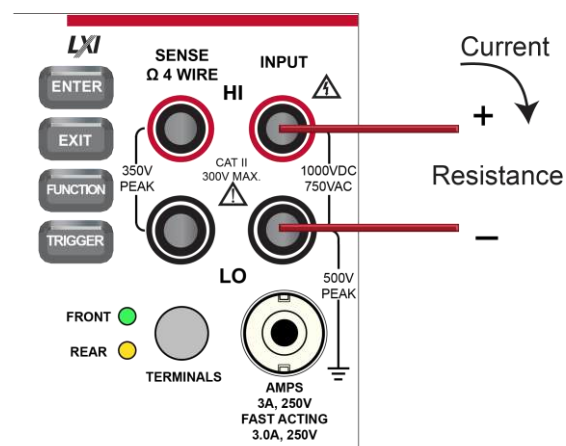
Always connect the sense lines as close as possible to the device under test.

Figure 68: Four-wire DUT connections to the front panel



2-wire resistance measure connections

Figure 69: Front-panel connections: 2-wire resistance measurement



Measure 2-wire resistance using the front panel

To make a 2-wire resistance measurement using the front panel:

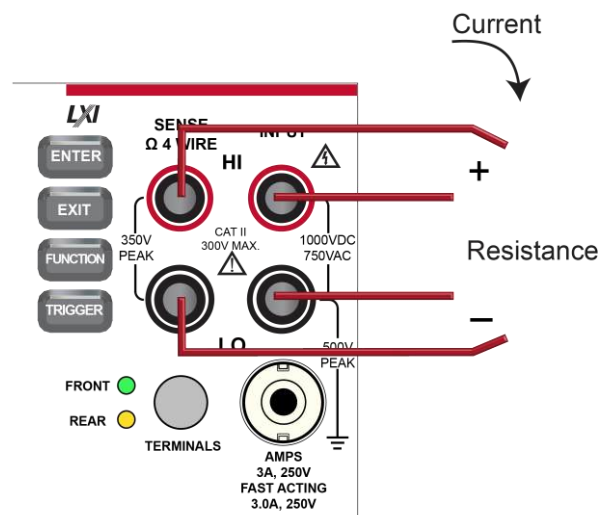
1. Make the connections as shown in [2-wire resistance measure connections](#) (on page 4-18).
2. Press the **FUNCTION** key.
3. Select **2W Res.**
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

Settings available for 2-wire resistance measurements

See [2-wire resistance measure settings](#) (on page 3-41) for settings that are available when you are making 2-wire resistance measurements.

4-wire resistance measure connections

Figure 70: Front-panel connections: 4-wire resistance measurement



Measure 4-wire resistance using the front panel

To make a 4-wire resistance measurement using the front panel:

1. Make the connections as shown in [4-wire resistance measure connections](#) (on page 4-19).
2. Press the **FUNCTION** key.
3. Select **4W Res.**
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for 4-wire resistance measurements

See [4-wire resistance measure settings](#) (on page 3-42) for settings that are available when you are making 4-wire resistance measurements.

Offset-compensated ohms

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

For 4-wire resistance measurements, when offset compensation is enabled, the measure range is limited to a maximum of 10 k Ω . When Auto is selected, the instrument automatically turns offset compensation on or off as appropriate for the selected range.

For 2-wire resistance measurements, offset compensation is always set to off.

For temperature measurements, offset compensation is only available when the transducer type is set to an RTD option.

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

Offset-compensated ohm calculations

NOTE

Instrument operations, including offset-compensated ohms, are performed on the input signal in a sequential manner.

For a normal resistance measurement, the DAQ6510 sources a current (I) and measures the voltage (V). The resistance (R) is then calculated as ($R = V/I$) and the reading is displayed.

For offset-compensated ohms, two measurements are performed: One normal resistance measurement and one measurement using the lowest current source setting.

The offset-compensated ohms reading is then calculated as follows:

$$\text{Offset-compensated } \Omega = \frac{\Delta V}{\Delta I}$$

where:

$$\Delta V = V_2 - V_1$$

$$\Delta I = I_2 - I_1$$

- V_1 is the voltage measurement with the current source at its normal level.
- V_2 is the voltage measurement using the lowest current source setting.
- I_1 is the current measurement with the source set to a specific level.
- I_2 is the current measurement with the source set to zero.

This 2-point measurement process and reading calculation eliminates the resistance contributed by the presence of V_{EMF} .

Continuity measurements

This section describes how you can set up continuity measurements from the front panel.

The DAQ6510 can test continuity using the 2-wire 1 k Ω range with a user-selected threshold resistance level. When the measured circuit is below the set threshold level, the instrument displays the resistance readings. When the measured circuit is above the threshold level, the instrument displays the message OPEN.

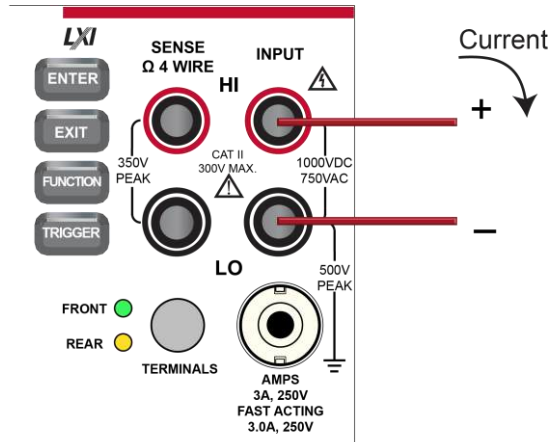
The continuity function does not support relative offset. Use the [mx+b](#) (on page 4-59) calculation, with b as an offset, to compensate for cable resistance.

NOTE

The reading rate for continuity is always set to 0.006 power line cycles.

Continuity measure connections

Figure 71: Front-panel connections: Continuity measurement



Measure continuity using the front panel

To make a continuity measurement using the front panel:

1. Make the connections as shown in [Continuity measure connections](#) (on page 4-22).
2. Press the **FUNCTION** key.
3. Select **Continuity**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for continuity measurements

See [Continuity measure settings](#) (on page 3-43) for settings that are available when you are making continuity measurements.

Frequency measurements

This section describes how you can set up frequency measurements from the front panel. Frequency measurements are only applicable to voltage signals.

Frequency and period support fixed and autorange threshold ranging, with a range of 100 mV to 750 V. Ranges are scaled to root-mean-square (RMS) sine wave voltages.

When autorange is selected, there are two measurement phases, measure AC voltage and measure frequency or period. When the AC voltage is measured, the amplitude is measured and the appropriate range is selected to ensure 11 percent to 110 percent signal scaling. In the second phase, the frequency or period is measured.

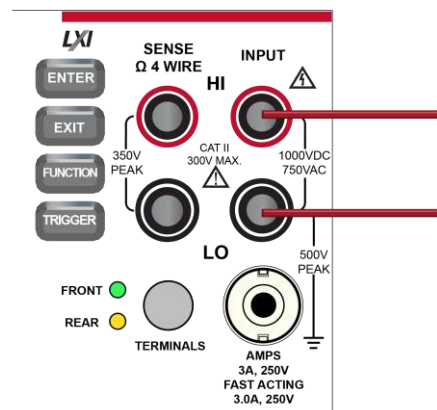
Frequency and period are specified for square wave inputs. The input signal must be more than 10 percent of the AC voltage range. If the input is less than 20 mV and measured on the 100 mV range, the frequency must be more than 10 Hz. For sine wave inputs, the input frequency must be more than 100 Hz.

CAUTION

Do not apply more than 1000 VDC between INPUT HI and LO. Failure to observe this caution may result in instrument damage.

Frequency measure connections

Figure 72: Front-panel connections: Frequency measurement



Measure frequency using the front panel

To make a frequency measurement using the front panel:

1. Make the connections as shown in [Frequency measure connections](#) (on page 4-23).
2. Press the **FUNCTION** key.
3. Select **Frequency**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for frequency measurements

See [Frequency measure settings](#) (on page 3-43) for settings that are available when you are making frequency measurements.

Period measurements

This section describes how you can set up period measurements from the front panel.

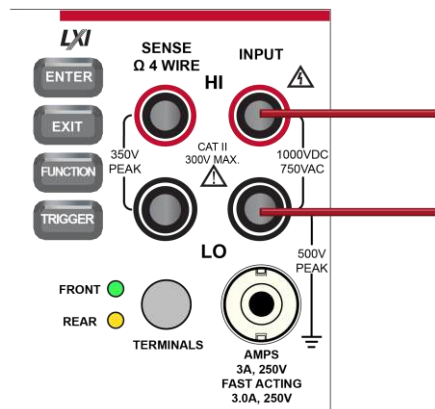
Period measurements are only applicable to voltage signals.

CAUTION

Do not apply more than 1000 VDC between INPUT HI and LO. Failure to observe this caution may result in instrument damage.

Period measure connections

Figure 73: Front-panel connections: Period measurement



Measure the period using the front panel

To make a period measurement using the front panel:

1. Make the connections as shown in [Period measure connections](#) (on page 4-24).
2. Press the **FUNCTION** key.
3. Select **Period**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for period measurements

See [Period measure settings](#) (on page 3-44) for settings that are available when you are making period measurements.

Diode measurements

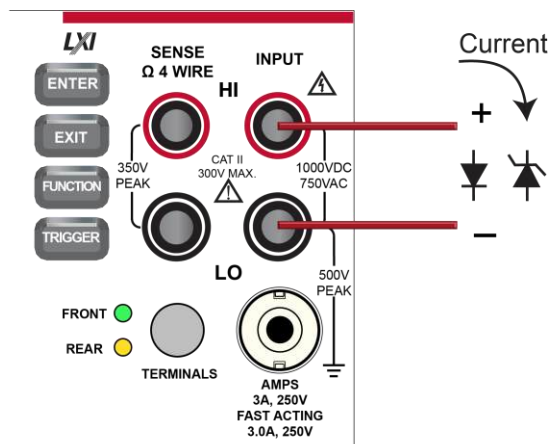
With a DAQ6510, you can measure the forward voltage drop of general-purpose diodes and the Zener voltage of Zener diodes. You can measure the forward voltage drop of a diode on the 10 V range with a constant test current (bias level). You can select a bias level of 10 μ A, 100 μ A, 1 mA, or 10 mA.

CAUTION

Do not apply more than 1000 VDC between INPUT HI and LO. Failure to observe this caution may result in instrument damage.

Diode measure connections

Figure 74: Front-panel connections: Diode measurement



Measure diode forward bias using the front panel

To make a diode measurement using the front panel:

1. Make the connections as shown in [Diode measure connections](#) (on page 4-26).
2. Press the **FUNCTION** key.
3. Select **Diode**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for diode measurements

See [Diode measure settings](#) (on page 3-45) for settings that are available when you are making diode measurements.

Temperature measurements

This section describes how to set up temperature measurements. You can measure temperature using thermoelectric transducers including thermocouples, thermistors, and resistance temperature detectors (RTDs).

Temperature measure connections

Figure 75: 2-wire thermistor connections

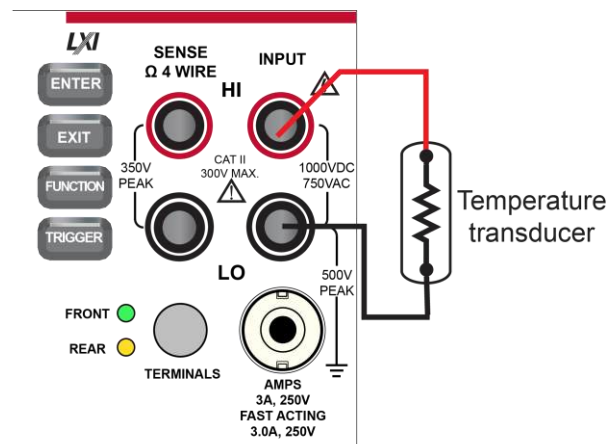
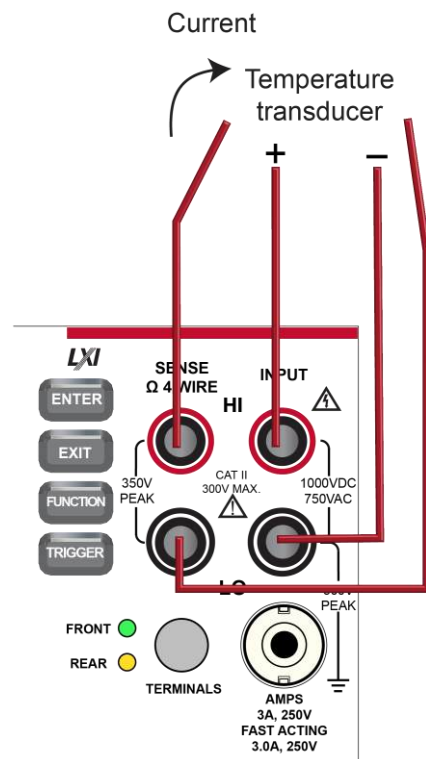


Figure 76: 4-wire RTD measurement



DAQ6510 front panel

Measure temperature using the front panel

To make a temperature measurement using the front panel:

1. Make the connections as shown in [Temperature measure connections](#) (on page 4-27).
2. Press the **FUNCTION** key.
3. Select **Temperature**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Temperature transducer types

You can use thermocouples, thermistors, 2-wire RTDs, 3-wire RTDs, and 4-wire RTDs with the DAQ6510.

For thermocouples, temperature measurement range depends on which type of thermocouple is being used. Thermocouple types B, E, J, K, N, R, S, and T are supported.

Simulated, internal, and external thermocouple reference junction types are supported by the DAQ6510. For more information, see [Reference junctions](#) (on page 4-88).

The thermistor types 2252 Ω , 5000 Ω , and 10,000 Ω are supported.

NOTE

Curve-fitting constants are used in the equation to calculate thermistor temperature. The thermistor manufacturer's specified curve fitting may not be the same as the ones used by the DAQ6510.

The DAQ6510 supports the following RTD types:

- PT100
- D100
- F100
- PT385
- PT3916

You can also select the user type. When the user type is selected, you can define the alpha, beta, delta, and zero values of the RTD.

For 2-wire RTD measurements, the HI and LO input terminals are used to measure temperature. The DAQ6510 makes a 2-wire resistance measurement and calculates the temperature based on the measured value and the selected type of RTD. Two-wire RTD measurements are less accurate than 3-wire and 4-wire RTD measurements because there is no compensation for resistance of the test leads, but two-wire measurements provide faster reading rates.

For 3-wire RTD measurements, the HI, LO, and SENSE LO input terminals are used to measure temperature. The SENSE LO terminal senses lead resistance and properly compensates the resistance measurement before converting to temperature. The accuracy for 3-wire RTD is within less than a 0.1 Ω lead resistance mismatch for INPUT HI and INPUT LO. Add 0.25 $^{\circ}\text{C}$ per 0.1 Ω of HI-LO lead resistance mismatch.

For 4-wire RTD measurements, by default, the DAQ6510 measures temperature with offset-compensated ohms and open lead detection enabled. This provides the most accurate and reliable method to measure the low resistance of the RTD. For faster RTD measurements when the most accurate measurements are not required, you can disable offset compensation and open lead detection for 3-wire and 4-wire RTD measurements.

Settings available for temperature measurements

See [Temperature measure settings](#) (on page 3-46) for settings that are available when you are making temperature measurements.

Capacitance measurements

With a DAQ6510, you can measure capacitance.

The capacitance function sources a constant I_{TEST} current through the device under test (DUT) while measuring voltage (dV) in a fixed time interval (dt). The capacitance measurement is:

$$I_{TEST} * dt / dV$$

Capacitance measurements have two measurement phases: Discharge and charge. During the discharge phase, the DUT is connected through an internal 13 mA current source and discharged to approximately 0 V. In the charge phase, the I_{TEST} is sourced while measuring the voltage. If the voltage on the DUT exceeds $2.8 \text{ V} \pm 10$ percent, the I_{TEST} is halted and the voltage is held until the discharge phase. If the voltage is less than 2.8 V, the resultant capacitance measurement is calculated.

Capacitance supports 1 nF to 100 μF ranges. Each range measures from 0 percent to 120 percent full scale. Reading rates vary based on range and the percent of full scale.

The 13 mA discharge and I_{TEST} currents are protected to 1000 V.

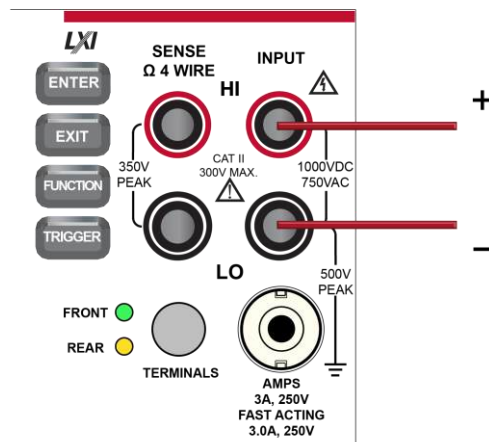
Capacitance has a fixed aperture time.

CAUTION

Do not apply more than 1000 VDC between INPUT HI and LO. Failure to observe this caution may result in instrument damage.

Capacitance measure connections

Figure 77: Front-panel connections: Capacitor measurement



Measure capacitance using the front panel

To make a capacitance measurement using the front panel:

1. Make the connections as shown in [Capacitance measure connections](#) (on page 4-30).
2. Press the **FUNCTION** key.
3. Select **Capacitance**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for capacitance measurements

See [Capacitance measure settings](#) (on page 3-47) for settings that are available when you are making capacitance measurements.

DC voltage ratio measurements

The DC voltage ratio function calculates the ratio between the measure input (numerator) and the reference voltage (denominator). This function can be useful when comparing one or more voltages to a single voltage. Only DC voltages can be compared.

The SENSE terminals are used as the reference voltage (V_s). The SENSE terminals can measure DC volts in the 100 mV, 1 V, and 10 V ranges.

The INPUT terminals provide the voltage (V_i) to be compared against the reference voltage. They can measure DC volts in the 100 mV, 1 V, 10 V, 100 V, and 1000 V ranges.

The ratio is calculated as:

$$\text{Ratio} = \frac{V_{\text{input}} - V_{\text{input\_rel}}}{V_{\text{sense}} - V_{\text{sense\_rel}}}$$

CAUTION

SENSE HI and LO must be referenced to INPUT LO.

SENSE HI must not exceed 125 percent, referenced to INPUT LO, of the selected sense range.

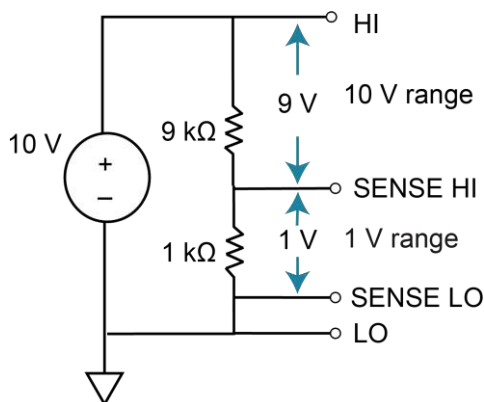
If you apply a relative offset value when using the DC voltage ratio function, the instrument removes the voltage on each terminal before the ratio calculation, which can result in unacceptable results. To set the behavior of relative offset with the DC voltage ratio function, use the SCPI command [\[:SENSe\[1\]\]:<function>:RElative:METHod](#) (on page 12-154) or the TSP command [dmm.measure.rel.method](#) (on page 14-287). This setting is not available using the front panel.

NOTE

To access the extra value in the reading buffer, the reading buffer style must be set to full. The extra value is available through the front panel in the Reading Details, through the SCPI command [:TRACe:DATA?](#) (on page 12-237), and through the TSP command [bufferVar.extravalues](#) (on page 14-48), [bufferVar.extraformattedvalues](#) (on page 14-46), and [bufferVar.extravalueunits](#) (on page 14-49). Refer to [Creating buffers](#) (on page 6-6) for information on setting the reading buffer style to full.

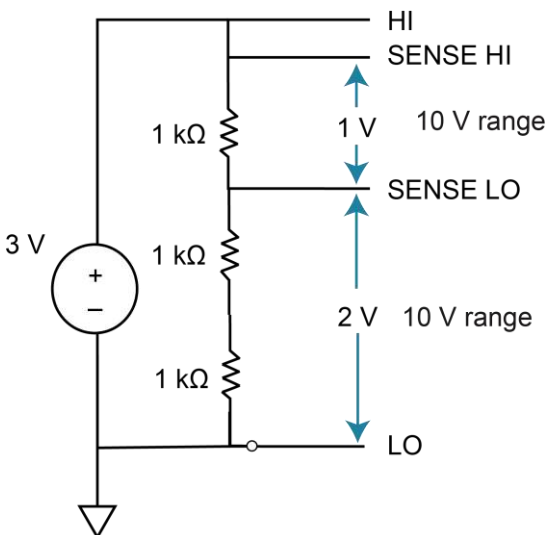
For example, if you have a 9 k Ω /1 k Ω resistive network, connect a 10 V source across the network. Connect measurement input HI and LO across the total 9 k Ω /1 k Ω resistive network and select the 10 V measure range. Connect Sense HI and LO across the 1 k Ω portion of the network and select the 100 mV range. The ratio measurement is approximately 10.00000.

Figure 78: DCV ratio 9 k Ω /1 k Ω resistor network example



Another example is a 1 k Ω /1 k Ω /1 k Ω resistor network. If 3 V is applied across the total three 1 k Ω resistors and V_{SENSE} is applied across the first 1 k Ω resistor, set V_{INPUT} to the 10 V range and V_{SENSE} to the 1 V range. The ratio measurement is approximately 3.00000. If V_{SENSE} is set to the 1 V range, the ratio displays *Overflow*, with SENSE HI and SENSE LO terminals exceeding the 125 percent maximum reference to the LO terminals. The SENSE HI to LO is 3 V and SENSE LO to LO is 2 V.

Figure 79: DCV ratio 1 k Ω /1 k Ω /1 k Ω resistor network

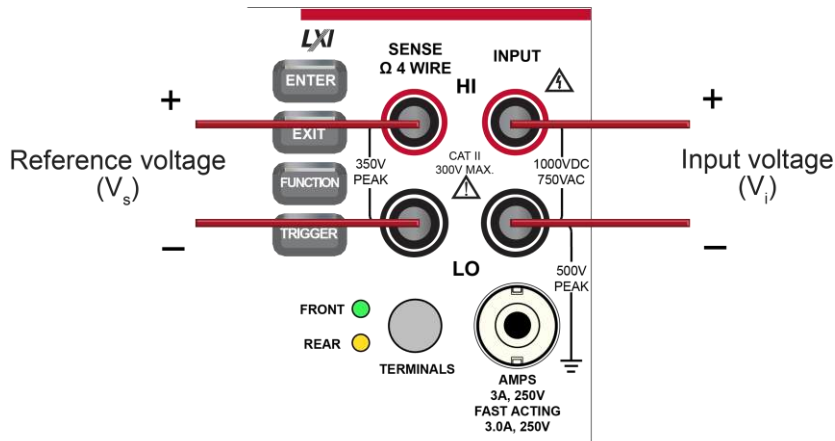


CAUTION

Do not apply more than 1000 VDC to the INPUT terminals or more than 350 V_{PEAK} to the SENSE terminals. Failure to heed this caution may result in instrument damage.

DC voltage ratio measure connections

Figure 80: Front-panel connections: DC voltage ratio measurement



Measure DC voltage ratio using the front panel

To make a DC voltage ratio measurement using the front panel:

1. Make the connections as shown in [DC voltage ratio measure connections](#) (on page 4-34).
2. Press the **FUNCTION** key.
3. Select **DCV Ratio**.
4. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for DC voltage ratio measurements

See [DC voltage ratio measure settings](#) (on page 3-47) for settings that are available when you are making DC voltage ratio measurements.

Digitize functions

The DAQ6510 digitize functions make fast, predictably spaced measurements. The speed, sensitivity, and bandwidth of the digitize functions allows you to make accurate voltage and current readings of fast signals, such as those associated with sensors, audio, medical devices, power line issues, and industrial processes. The digitize functions can provide 1,000,000 readings per second at $4\frac{1}{2}$ digit resolution. Digitize voltage and digitize current have separate internal signal paths that are optimized for fast response to signal changes.

The sample rate determines how often the readings are output by the digitize function. You can set it from 1000 to 1,000,000 readings per second.

The aperture determines the reading conversion time. This is when data is gathered to create the reading. You set the aperture time in 1 μ s intervals. If the aperture is more than 1 μ s, the consecutive 1 μ s readings are averaged to produce the reading.

The sample rate affects the available aperture settings. The maximum aperture is determined by $1/\text{sample rate}$ (rounded down to the nearest integer). The instrument automatically adjusts the aperture setting if the sample rate is changed to a rate that does not support the existing aperture setting. When this occurs, a warning message is generated that reports the new aperture setting.

The count is the number of times to make readings with the selected sample rate and aperture after a trigger is detected. In continuous mode, the instrument generates automatic triggers. In manual mode, a trigger is defined by pressing the TRIGGER key on the front panel. You can also set up other types of triggers. For more information on triggers, refer to [Triggering](#) (on page 8-18).

If you are using the TSP command language, the commands use a different syntax for measure and digitize commands. For example, the command to change the measure function range is:

```
dmm.measure.range = 100
```

The command to change the digitize range is:

```
dmm.digitize.range = 100
```

You can set the digitize voltage range from 100 mV to 1000 V. The digitize current range can be set from 100 μ A to 3 A.

Digitize functions do not support autorange, autozero, or autodelay.

NOTE

Digitize functions create a large amount of data quickly. Make sure the selected buffer is large enough for the expected data.

Digitize voltage measurements

The digitize voltage function makes accurate, predictably spaced voltage measurements.

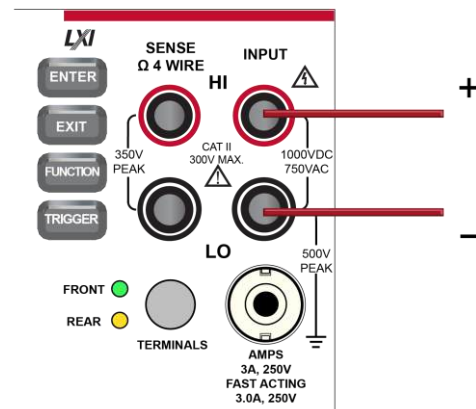
CAUTION

Do not apply more than 1000 VDC between INPUT HI and LO. Failure to observe this caution may result in instrument damage.

Digitize voltage measure connections

The connections for voltage measurements are shown in the following graphic.

Figure 81: Front-panel connections: Digitize voltage measurement



Measure with digitize voltage using the front panel

To make a digitize voltage measurement using the front panel:

1. Make the connections as shown in [Digitize voltage measure connections](#) (on page 4-36).
2. Press the **FUNCTION** key.
3. Select the **Digitize Functions** tab.
4. Select **Digitize Voltage**.
5. Press the **MENU** key.
6. Under Measure, select **Settings**.
7. Select the settings for your application. For descriptions of the options, refer to [Digitize Voltage measure settings](#) (on page 3-48).
8. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements display on the front panel.

Settings available for digitize voltage measurements

See [Digitize voltage measure settings](#) (on page 3-48) for settings that are available when you are digitizing voltage measurements.

Show voltage readings in decibels

You can show DC or AC voltage in decibels (dB), which compresses a large range of measurements into a much smaller scope. The relationship between dB and voltage is defined by the following equation:

$$\text{dB} = 20 \log \left| \frac{V_{\text{in}}}{V_{\text{ref}}} \right|$$

Where:

- V_{in} is the DC or AC input signal
- V_{ref} is the specified voltage reference level

If a relative offset value is in effect when dB is selected, the value is converted to dB, and then relative offset is applied to the dB value. If relative offset is applied after dB has been selected, dB has relative offset applied to it.

NOTE

The largest negative value of dB is –180 dB. This accommodates a ratio of $V_{\text{in}} = 1 \mu\text{V}$ and $V_{\text{ref}} = 1000 \text{ V}$.

Show voltage readings in decibel-milliwatts (dBm)

Decibel-milliwatts (dBm) is used to express an absolute value of power.

To calculate dBm, use the formula:

$$\text{dBm} = 10 \log \frac{(V_{\text{IN}})^2}{R_{\text{REF}} \cdot 1 \text{ mW}}$$

Where:

- V_{IN} is the voltage in
- R_{REF} is the reference impedance

Digitize current measurements

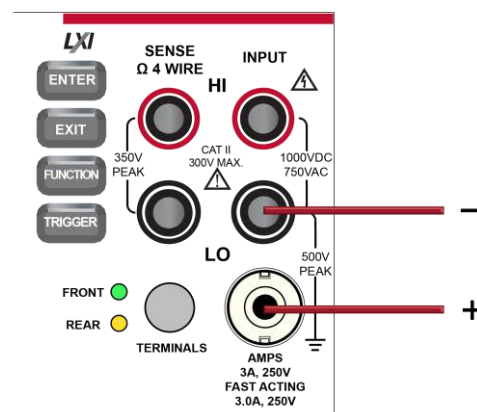
The digitize current function makes accurate, predictably spaced current measurements.

CAUTION

Do not apply more than 500 V_{PEAK} between INPUT LO and the AMPS input. Failure to observe this caution may result in instrument damage.

Digitize current measure connections

Figure 82: Front-panel connections: Digitize current measurement



Measure with digitize current using the front panel

To make a digitize current measurement using the front panel:

1. Make the connections as shown in [Digitize current measure connections](#) (on page 4-39).
2. Press the **FUNCTION** key.
3. Select the **Digitize Functions** tab.
4. Select **Digitize Current**.
5. Press the **MENU** key.
6. Under Measure, select **Settings**.
7. Select the settings for your application. For descriptions of the options, refer to [Digitize Current measure settings](#) (on page 3-48).
8. Press the **TRIGGER** key for two seconds and verify that the instrument is set to Continuous Measurement.

The measurements start displaying on the front panel.

Settings available for digitize current measurements

See [Digitize current measure settings](#) (on page 3-48) for settings that are available when you are digitizing current measurements.

Digitizing aperture and sample rate

In most cases, you will get good results if you leave the aperture at the default setting of automatic. When Auto is selected, the instrument makes as many measurements as possible in the sample period. When it is set automatically, aperture is set to 1 million per second (rounded down to the nearest integer).

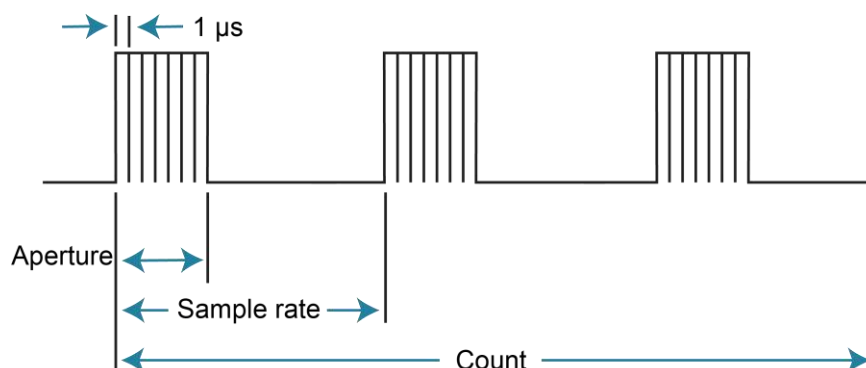
You may want to set a manual aperture if you need an aperture that contains higher than one discrete 1 μ s averaged reading.

Although the maximum sample rate is 1 million samples per second, the input filtering of the analog/digital (A/D) converter is set at a 3 dB corner point of slightly greater than 350 kHz to prevent aliasing. Therefore, a 350 kHz or higher voltage input is attenuated by a factor of 0.707. For dynamic signals, this attenuation could cause attenuated readings. Consult the specifications for detail.

Input frequencies above 500 kHz are occasionally prone to the signal processing problem of aliasing.

The following figure shows the relationship between the aperture, sample rate, and count.

Figure 83: Digitize aperture, sample rate, and count



For large count (more than 8,000,000) and sample rate values (more than 150,000), data may be lost. Adjust one of the values to a lower level.

Display results of two measure functions

The DAQ6510 allows you to make and display two measurements from different functions. The measurements are displayed on the front panel and stored in the reading buffers.

The measurements from the secondary function are automatically saved to `defbuffer2`. If the active buffer is set to `defbuffer2`, you cannot select the Secondary Measure function. If you cannot select the measurement function, verify that the active buffer is set to `defbuffer1` or a user-defined buffer.

To access the dual measurement capability, swipe the lower half of the home screen to the SECONDARY swipe screen. This feature is only available from the front panel of the instrument when the instrument is set to the measurement method Continuous Measurement or Manual Trigger Mode.

The secondary measurement function is only available when the front terminals are selected. Both measurement functions must be using the same set of terminals.

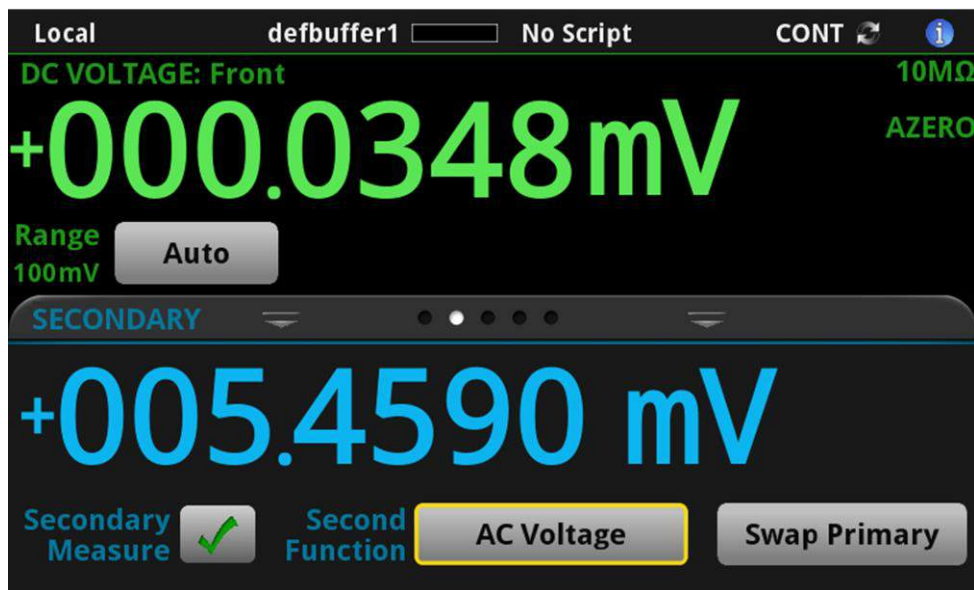
The following measurements can be paired without impacting the specifications of either measurement. While you can use other combinations, you might not achieve specifications for both measurements at the same time. For each pair in the following table, either function can be the primary measurement.

Measurement function	Paired measurement function
DCV	ACV
DCV	DCV Ratio
DCV	Temperature (thermocouple only)
DCI	ACI
ACV	Frequency
ACV	Period
ACV	Frequency
ACI	Period
ACI	Frequency
Frequency	Period
2W Resistance	Capacitance
2W Resistance	Continuity
2W Resistance	Temperature (RTD or thermistor)
2W Resistance	4W Resistance
4W Resistance	Continuity
4W Resistance	Temperature (RTD or thermistor)

NOTE

Depending on the selected functions, a relay may click when the instrument switches between the measurement types. Leaving secondary measurements on for extended periods may shorten the life of the relays.

Figure 84: SECONDARY swipe screen



Making secondary measurements

When you are using the secondary measurements feature, any settings that you change from the front panel of the instrument affect the primary function (the function shown at the top of the Secondary swipe screen). To change settings for the secondary function (the function shown at the bottom of the Secondary swipe screen), you need to swap the functions. Select the **Swap Primary** button to switch the primary and secondary functions. Changes made for a particular measure function while in the primary function remain set for that measure function until specifically changed.

Measurements are stored in separate reading buffers. By default, the primary measurements are stored in `defbuffer1` and secondary measurements are stored in `defbuffer2`. For the primary measurement, you can change the reading buffer by selecting the buffer and making it active. Refer to [Using the front panel to select a reading buffer](#) (on page 6-18) for detail. You cannot change the reading buffer for the secondary measurement. If you change the active buffer, clear `defbuffer2` before triggering a new measurement. If you do not clear `defbuffer2`, it remains aligned with the previous active buffer, even if that buffer was deleted.

When graphing secondary measurements, the timestamp for the buffer that contains the data for the secondary measurement is tracked relative to the timestamp of the buffer that is storing the primary measurement. For example, if secondary measurements are enabled 5 seconds after the primary measurements, then plotting only the secondary buffer shows readings starting at 5 seconds.

If `defbuffer2` contains readings from previous measurements that were not aligned with the active buffer and a new secondary measurement is started, the time will not align with the buffer that contains the primary measurement data.

Secondary measurements are not available for use with the trigger model.

To make secondary measurements:

1. Make connections to the instrument appropriate to both types of measurements. Refer to [Measurement overview](#) (on page 4-4) for connection information.
2. Swipe to the **SECONDARY** swipe screen.
3. Set up the primary function as needed.
4. Hold the **TRIGGER** key for 2 seconds and select **Continuous Measurement** or **Manual Trigger Mode**.
5. Select **Second Function** to select the secondary function.
6. Select **Secondary Measure**.

7. If you selected Continuous Measurements, measurements for both functions begin. If you selected Manual Trigger Mode, measurements are made when you press the **TRIGGER** key.
8. If you need to change settings for the secondary function, select **Swap Primary**. Make the settings as needed, then select **Swap Primary** again.

Displayed measurements

When you make measurements, the instrument may perform operations on the measured values that affect what you see on the display and the measurements that are stored in the buffer.

The operations that can affect the measurement display are:

- Filtering
- Relative offset
- Math operations
- Limit tests

If none of these operations is set, the value that is displayed on the front panel is the actual measurement reading.

If any one of these operations is set, the value that is displayed is the measurement reading with these operations applied. The operations are applied in the order shown above.

For example, if you made a measurement and had a relative offset and limit tests active, the measured value would have the relative offset applied, then have limit test results applied.

For additional detail on the order of operations, see [Order of operations](#) (on page 4-91).

Using Quickset

You can specify a function and adjust the performance of your DAQ6510 using the options on the Quickset menu.

The Function option changes the measurement function. The functions are the same as those that are available using the FUNCTION key and FUNCTIONS swipe screen.

Using the Performance slider

Use the Performance slider to adjust for performance (resolution versus speed).

When you adjust the Performance slider, the instrument changes settings based on where you position the slider. As you increase reading speed, you lower the amount of resolution. As you increase resolution, you decrease the speed. These settings take effect the next time measurements are made.

NOTE

To see which settings are adjusted, you can set the Command setting of the Event Log to On. When command logging is on, each setting made by the Performance slider is listed as an Information event in the Event Log.

If the instrument is set to the DC voltage, DC current, digitize voltage, or digitize current function, changing the speed may change the function from the DC voltage or DC current function to the digitize voltage or digitize current function and vice versa.

When the temperature function is selected, the readings per second are shown as a range to accommodate the various transducer types.

Store settings for functions regardless of active state

When you are using the front panel or TSP commands, changes to settings affect the function that is presently selected.

If you need to set up functions that are not selected, you can use the `dmm.measure.setattribute` command. This command applies settings to a specific function, whether or not the function is selected. If you are changing functions during a test and want to improve the speed of the test, this eliminates the time needed to change the settings for each function during the test.

For example, the following set of commands sets up the DC Current function. When you select the DC Current function, these settings are active immediately.

```
-- Active measure function is DC Voltage.  
-- Configure DC Current settings without changing the active function.  
dmm.measure.setattribute(dmm.FUNC_DC_CURRENT, dmm.ATTR_MEAS_RANGE, 35e-6)  
dmm.measure.setattribute(dmm.FUNC_DC_CURRENT, dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_5_5)  
dmm.measure.setattribute(dmm.FUNC_DC_CURRENT, dmm.ATTR_MEAS_NPLC, 0.5)
```

Measurement methods

Triggers are signals that instruct the instrument to make a measurement. You can set the DAQ6510 to use the following triggering measurement methods:

- **Continuous measurement:** The instrument is making measurements continuously.
- **Manual trigger mode:** Press the front-panel TRIGGER key to initiate a single measurement.
- **Trigger model:** The instrument makes measurements according to the settings of the trigger model. To select this method, a trigger model must be set up. Select Initiate Trigger Model to start the trigger model or Abort Trigger Model to stop a trigger model that is presently running.

Continuous measurement triggering

When you select the continuous measurement method, the instrument makes measurements continuously.

The continuous measurement method is only available when you are controlling the instrument locally (through the front panel).

The instrument stores the readings in the active reading buffer. See [Reading buffers](#) (on page 6-1) for detail on the buffer options that are available.

If you press the front-panel TRIGGER key when the instrument is set to the continuous measurement method, measurements are not made. Instead, a dialog box is displayed that asks if you want to change the measurement method.

Trigger key triggering

When you select the Manual Trigger Mode from the DAQ6510 front-panel, the instrument only makes a measurement when you press the front-panel TRIGGER key.

The instrument stores the readings in the active reading buffer. See [Reading buffers](#) (on page 6-1) for detail on the buffer options that are available.

Trigger model triggering

When you select the trigger model measurement method, the instrument uses a trigger model to control the sequence in which measurements occur. The DAQ6510 trigger model is flexible, allowing you to control as much or as little as needed for your measurement application.

When you are remotely controlling the instrument, the trigger model measure method is automatically selected. In addition, you can view different buffers from the front panel, but the actual buffer that is used is defined by the remote commands.

For detail on the trigger model, see [Trigger model](#) (on page 8-35).

Switching between measurement methods

The measurement methods that are available to you depend on how you are controlling the instrument.

If you are using the front panel to control the instrument, you can choose any of the measurement methods.

If you are using a remote interface to control the instrument, you can only use the trigger model measurement method. When you switch to a remote interface, the trigger model measurement method is automatically selected. If you switch from remote control to front-panel control, the trigger model measurement method remains selected.

If you are running a script, the instrument automatically switches to the trigger model measurement method.

Using the front panel:

1. Press the front-panel **TRIGGER** key for 2 seconds. A dialog box displays the available trigger methods. The presently selected method is highlighted.
2. Select the method you want to use.
3. If the instrument is in remote control, the instrument displays a confirmation dialog box. Select **Yes** to change to local control.

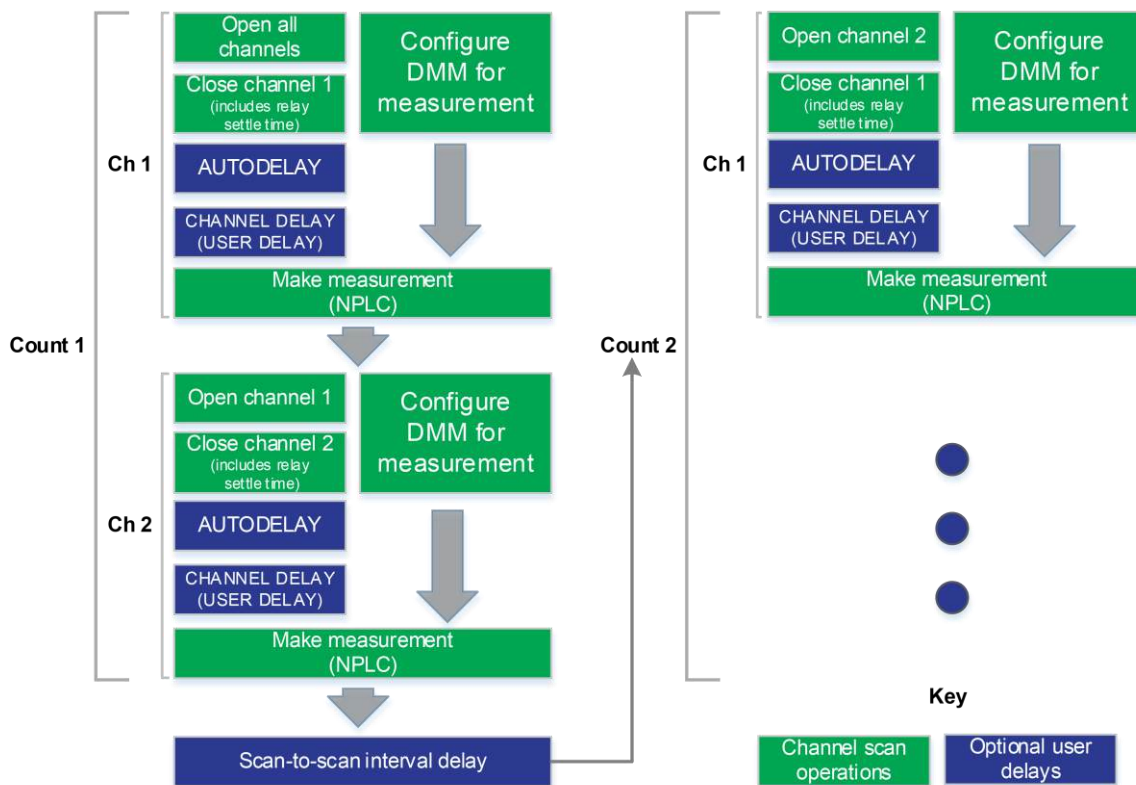
Auto Delay

Auto Delay applies a wait period at the end of a function change, range change, and other measure-related settings. When you change functions or ranges, an additional autodelay is applied to allow for settling time. The delay allows cables or internal DMM circuitry to settle for best measurement accuracy.

- When autodelay is disabled, no wait time is applied.
- When autodelay is enabled, a measurement is not made until immediately after the autodelay period has expired.

The following diagram illustrates a two-channel scan using autodelay with a scan count of greater than one. The scan begins with all channels open, then channel 1 is closed, and a relay settling time is added. At the same time, the DMM is configured for the measurement.

The first measurement is made at this time unless you specify an autodelay. Depending on the measurement function you have selected, the optional autodelay is inserted into the measurement time, followed by an optional channel delay. When the delays are complete, the measurement is made. At the end of the first scan count, an optional scan-to-scan interval delay is applied, and the scan continues to the next count.

Figure 85: Example: Two-channel scan list with a scan count greater than 1

Autodelay times for each function are provided in the following topics.

NOTE

The following times represent a DAQ6510 with a Model 7700 20-Channel Differential Multiplexer Module installed. These times are also accurate for the following Keithley Instruments switching modules:

- 7701
- 7702
- 7706
- 7707
- 7708
- 7709

For other Keithley Instruments switching modules, see the applicable module specification.

To set autodelay for the selected function from the front panel:

1. Press the **MENU** key.
2. Under Measure, select **Settings**.
3. Set Auto Delay to **On** to include a delay or **Off** to remove the delay.

To set autodelay using SCPI commands:

Refer to [\[:SENSe\[1\]\]:<function>:DELAy:AUTO?](#) (on page 12-138)

To set autodelay using TSP commands:

Refer to [dmm.measure.autodelay](#) (on page 14-205)

Voltage autodelay times

The following table provides times for autodelay for the DAQ6510 voltage functions.

Function	Detector bandwidth (Hz)	Range and delays						
DC volts	Not applicable	Range	100 mV	1 V	10 V	100 V	1000 V	Autorange on
		Autodelay	1.5 ms	1.5 ms	0 ms	1 ms	1.5 m	1.5 ms
AC volts	Not applicable	Range	100 mV	1 V	10 V	100 V	1000 V	Autorange on
	3, 30, or 300	Autodelay	400 ms					

Current autodelay times

The following tables provide times for autodelay for the DAQ6510 current functions.

Function	Range and delays								
DC current	Range	10 μ A	100 μ A	1 mA	10 mA	100 mA	1 A	3 A	Autorange on
	Autodelay	0 ms							

Function	Detector bandwidth (Hz)	Range and delays							
AC current	Not applicable	Range	100 μ A	1 mA	10 mA	100 mA	1 A	3 A	Autorange on
	3, 30, or 300	Autodelay	400 ms						

Resistance autodelay times

The following tables provide times for autodelay for the DAQ6510 resistance functions.

For continuity, the range is 1 k Ω with an autodelay of 3 ms.

Function	Range and delays									
2-wire ohm	Range	10 Ω	100 Ω	1 k Ω	10 k Ω	100 k Ω	1 M Ω	10 M Ω	100 M Ω	Autorange on
	Autodelay	9 ms	1 ms	1 ms	1 ms	5 ms	25 ms	115 ms	140 ms	25 ms

Function	Range and delays										
4-wire ohm	Range	1 Ω	10 Ω	100 Ω	1 k Ω	10 k Ω	100 k Ω	1 M Ω	10 M Ω	100 M Ω	Autorange on
	Autodelay	11.5 ms	9 ms	2 ms	2 ms	2 ms	5 ms	40 ms	180 ms	275 ms	40 ms

Frequency and period autodelay times

The following table provides times for autodelay for the DAQ6510 frequency and period functions.

Function	Ranges and delays						
Frequency and period	Range	100 mV	1 V	10 V	100 V	750 V	Autorange on
	Autodelay	1 ms					

Temperature autodelay and autorange times

The following table provides times for autodelay for the DAQ6510 thermistor and RTD temperature functions.

When thermocouple is selected, the range is 100 mV, with an autodelay of 1 ms.

Function	Range and delays							
Capacitance	Range	1 nF	10 nF	100 nF	1 μ F	100 μ F	1 mF	Autorange on
	Autodelay	1 ms						

Function	Range and delays					
Thermistor	Range	1 Ω to 100 Ω	1 k Ω	10 k Ω	100 k Ω	Autorange on
	Autodelay	1 ms				
3-wire and 4-wire RTD	Range	100 Ω	1 k Ω	10 k Ω	100 k Ω	Autorange on
	Autodelay	1 ms				

Capacitance autodelay times

Autodelay is not applied to DAQ6510 capacitance measurements.

Diode autodelay times

The diode autodelay time for all ranges, including autorange, 1 ms.

Detector bandwidth

You can select the detector bandwidth for AC voltage and AC current measurements. You can select 3 Hz, 30 Hz, or 300 Hz.

When you select the 3 Hz bandwidth, the signal goes through an analog root-mean-square (RMS) converter. The output of the RMS converter goes to a fast (1 kHz) sampling analog-to-digital converter and the RMS value is calculated from 1200 digitized samples (1.2 seconds).

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

When you select the 300 Hz bandwidth, the output of the analog RMS converter (nearly pure DC at these frequencies) is measured at an integration rate of 16.6 ms (1 power line cycle). You can set the integration rate from 8.333 μ s to 0.25 ms (60 Hz) and 10 μ s to 0.24 ms (50 Hz).

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

Automatic reference measurements

To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The DAQ6510 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks the reference measurements after every signal measurement. This can cause some measurements to take longer than normal.

This additional time can cause problems in test sequences in which measurement timing is critical. To avoid the time that is needed for the reference measurements, you can disable the automatic reference measurements.

When automatic reference measurements are turned off, the instrument may gradually drift out of specification. To prevent inaccurate readings, you can use autozero once to update the autozero information.

Setting autozero

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

The reference setting is stored with the selected measure function.

To set autozero using the front panel:

1. Press the **FUNCTION** key.
2. Select the measure function.
3. Press the **MENU** key.
4. Under Measure, select **Settings**.
5. For Auto Zero, select **On** or **Off**.
6. If Off is selected, you can select the **Once** option to send a one-time refresh.
7. Select **HOME** to return to the operating display.

To set autozero using SCPI commands, refer to the following commands:

- [\[:SENSe\[1\]\]:<function>:AZERo\[:STATe\]](#) (on page 12-133)
- [\[:SENSe\[1\]\]:AZERo:ONCE](#) (on page 12-177)

To set autozero using TSP commands, refer to the following commands:

- [dmm.measure.autozero.enable](#) (on page 14-207)
- [dmm.measure.autozero.once\(\)](#) (on page 14-211)

Ranges

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

You can allow the DAQ6510 to choose the range automatically or you can select a specific range.

Autorange selects the best range in which to measure the applied signal. If the measurement reaches 105 percent of the present range, the instrument changes the measurement range to the next higher range. The measurement range is changed when a measurement is made. Autorange is not available for the digitize functions.

When you select a specific range, the instrument remains at the value you selected. This option is intended to eliminate the time that is required by the instrument to automatically search for a range. When selecting a measure range, to ensure the best accuracy and resolution, use the lowest range possible that does not cause an overflow event. Note that when you select a fixed range, overrange conditions can occur.

If you set a specific measure range for a function, autorange is turned off for that function and remains off until you re-enable it.

NOTE

You need to set the measure function before setting the measure range. The range value is stored with the measure function.

Selecting the automatic measurement range

Using the front panel:

1. Press the **FUNCTION** key and select the function.
2. On the measure area of the home screen, select **Range**. The Measure Range dialog box is displayed.
3. Select **Auto**. The actual range is displayed to the left of the button.

Using a remote interface:

- SCPI commands: Refer to [\[:SENSe\[1\]\]:<function>:RANGe:AUTO](#) (on page 12-148).
- TSP commands: Refer to [dmm.measure.autorange](#) (on page 14-209).

Selecting a specific measure range

From the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. On the home screen, select **Range** in the measurement view area. The Measure Range dialog box is displayed.
3. Select the range. The selected value is displayed.

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

Using a remote interface:

- SCPI commands: Refer to [\[:SENSe\[1\]\]:<function>:RANGe\[:UPPer\]](#) (on page 12-149).
- TSP commands: Refer to [dmm.measure.range](#) (on page 14-275).

Relative offset

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

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

$$\text{Displayed value} = \text{Actual measured value} - \text{Relative offset value}$$

When a relative offset value is established for a measure function, the value is the same for all ranges for that measure function. For example, if 4 V is set as the relative offset value on the 100 V range, the relative offset value is also 4 V on the 1 V and 100 mV ranges.

On the front panel, when relative offset is enabled, the REL indicator to the right of the measured value is displayed.

A relative offset value is saved for each function. If you change the measure function, the relative offset value is changed to the setting for that measure function.

The relative offset is applied to the measurement before any math and limit test functions. For more information on the order in which operations are performed, see [Order of operations](#) (on page 4-91).

NOTE

You can perform the equivalent of relative offset manually by using the [mx+b](#) (on page 4-59) math function. Set m to 1 and b to the value of the offset.

Establishing a relative offset value

You can use the DAQ6510 to automatically determine the relative offset, or you can assign a specific relative offset value.

Automatically acquiring a relative offset value

When you automatically acquire a relative offset value, the DAQ6510:

- Makes a new measurement.
- Stores the measurement as the new relative offset level.

Before acquiring the offset, apply the signal that you want to offset the measurement by.

Using the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Select **Calculations**.
4. For Rel, select **Acquire**. The relative offset value is displayed to the right.

When you select **Acquire** from the front panel, Rel is automatically set to On unless an overflow reading is detected.

NOTE

You can also enable or disable the relative offset feature through the SETTINGS swipe screen Rel option.

To acquire a relative offset value for a channel:

1. Press the **HOME** key.
2. Swipe to the Channel swipe screen.
3. Select the channel.
4. Close the channel.
5. Swipe to the Settings Swipe screen.
6. Select **Rel**. If it is already selected, clear Rel and select it again. A relative offset value is acquired when you select Rel.

NOTE

You cannot acquire relative offset values for multiple channels.

Using a remote interface:

- SCPI commands: Refer to [\[:SENSe\[1\]\]:<function>:RELative:ACQuire](#) (on page 12-153) and [\[:SENSe\[1\]\]:<function>:RELative:STATe](#) (on page 12-155).
- TSP commands: Refer to [dmm.measure.rel.acquire\(\)](#) (on page 14-282) and [dmm.measure.rel.enable](#) (on page 14-283).

Setting a relative offset value

You can set a specific relative offset value using the front panel or remote commands.

Using the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Select **Calculations**.
4. For Rel, select **On**.
5. Select **Rel Value**.
6. Enter the value and select **OK**.

Using SCPI commands, send the commands:

```
:SENSe:FUNCTION "VOLTage"  
:SENSe:VOLTage:RELative <n>  
:SENSe:VOLTage:RELative:STATe ON
```

Where <n> is the amount of the offset.

To set the relative offset for another function, replace `VOLTage` with `CURRent` or `RESistance`. Refer to [\[:SENSe\[1\]\]:<function>:RELative](#) (on page 12-151) and [\[:SENSe\[1\]\]:<function>:RELative:STATe](#) (on page 12-155) for additional information.

Using TSP commands, send the commands:

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE  
dmm.measure.rel.level = relValue  
dmm.measure.rel.enable = dmm.ON
```

Where `relValue` is the relative offset value.

To set the relative offset for another function, replace `dmm.FUNC_DC_VOLTAGE` with `dmm.FUNC_DC_CURRENT` or `dmm.FUNC_RESISTANCE`. Refer to [dmm.measure.rel.level](#) (on page 14-284) and [dmm.measure.rel.enable](#) (on page 14-283) for additional information.

Calculations that you can apply to measurements

The DAQ6510 allows you to apply the following math operations to the measurement:

- $mx+b$
- percent
- reciprocal ($1/X$)

Math calculations are applied to the input signal after relative offset and before limit tests. For more detail on the order of operations, see [Order of operations](#) (on page 4-91).

Math operations apply to the selected measure function. If you change the measure function, the math operation for that function becomes active.

NOTE

Changing math functions does not clear the reading buffer, which can result in mixed units in the reading buffer. If you are graphing, this can cause ? to be displayed on the y-axis. Clear the reading buffer to remove the mixed units.

$mx+b$

The $mx+b$ math operation lets you manipulate normal display readings (x) mathematically based on the following calculation:

$$mx + b = y$$

Where:

- **m** is a user-defined constant for the scale factor
- **x** is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- **b** is a user-defined constant for the offset factor
- **y** is the displayed result

When the $mx+b$ math operation is active, the unit of measure for the front-panel readings is **X** and the MATH indicator is displayed to the right of the measurement. You cannot change this units designator.

Set the relative offset using mx+b

You can use the mx+b function to manually establish a relative offset value. To do this, set the scale factor (m) to 1 and set the offset (b) to the offset value. Each subsequent reading is the difference between the actual input and the offset value.

Percent

The percent math function displays measurements as percent deviation from a specified reference constant. The percent calculation is:

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

Where:

- **Percent** = The result
- **Input** = The measurement (if relative offset is being used, this is the relative offset value)
- **Reference** = The user-specified constant

The result of the percent calculation is positive when the input is more than the reference. The result is negative when the input is less than the reference.

When the percent operation is active, the unit of measure for the front-panel readings is % and the MATH indicator is displayed to the right of the measurement. You cannot change the unit designator.

Reciprocal (1/X)

You can set math operation to reciprocal to display the reciprocal of a reading.

The reciprocal is $1/X$, where X is the reading. If relative offset is on, the $1/X$ calculation uses the input signal with the relative offset applied.

Example:

Assume the normal displayed reading is $002.5000\ \Omega$. The reciprocal of resistance is conductance. When the reciprocal math function is enabled, the following conductance reading is displayed:

0.400000

When the reciprocal math operation is active, the unit of measure for the front-panel readings is $1/x$ and the MATH indicator is displayed to the right of the measurement. You cannot change this units designator.

Setting mx+b math operations

From the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Under Measure, select **Calculations**.
4. For Math, select **On**.
5. Select **Settings**.
6. For Math Format, select **mx+b**.
7. For m(Scalar), set the **m** value.
8. For b(Offset), set the **b** value.
9. Select **OK**.
10. Press the **HOME** key to view the measurement with the mx+b math format applied.

Using a remote interface:

- SCPI commands: Refer to [:CALCulate\[1\]:<function>:MATH:FORMat](#) (on page 12-29), [:CALCulate\[1\]:<function>:MATH:MMFactor](#) (on page 12-32), and [:CALCulate\[1\]:<function>:MATH:MBFactor](#) (on page 12-31).
- TSP commands: Refer to [dmm.measure.math.format](#) (on page 14-261), [dmm.measure.math.mxb.mfactor](#) (on page 14-265), and [dmm.measure.math.mxb.bfactor](#) (on page 14-263).

Setting percent math operations

From the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Under Measure, select **Calculations**.
4. Set Math to **On**.
5. Select **Settings**.
6. For Math Format, select **Percent**.
7. For Zero Reference, select the percent reference.
8. Select **OK**.
9. Press the **HOME** key to view the measurement with the percent math format applied.

Using a remote interface:

- SCPI commands: Refer to [:CALCulate\[1\]:<function>:MATH:FORMat](#) (on page 12-29) and [:CALCulate\[1\]:<function>:MATH:PERCent](#) (on page 12-34).
- TSP commands: Refer to [dmm.measure.math.format](#) (on page 14-261) and [dmm.measure.math.percent](#) (on page 14-267).

Setting reciprocal math operations

From the front panel:

1. Press the **FUNCTION** key and select the measure function.
2. Press the **MENU** key.
3. Under Measure, select **Calculations**.
4. For Math, select **On**.
5. Select **Settings**.
6. For Math Format, select **Reciprocal**.
7. Select **OK**.
8. Press the **HOME** key to view the measurement with the reciprocal math format applied.

Using a remote interface:

- SCPI commands: Refer to [:CALCulate\[1\]:<function>:MATH:FORMat](#) (on page 12-29).
- TSP commands: Refer to [dmm.measure.math.format](#) (on page 14-261).

Switching math on the SETTINGS swipe screen

Once you set the math operations settings for a measure function, you can turn the math function on or off on the SETTINGS swipe screen.

From the front panel:

1. Select **HOME**.
2. On the SETTINGS swipe screen, select or clear **Math** to enable or disable the selected math operation.
3. To change other math settings, select the calculations settings icon on the right side of the settings swipe screen to open the CALCULATION SETTINGS screen.

Filtering measurement data

Filters allow you to produce one averaged sample from a number of measurements. In situations where you have noise levels that fluctuate above and below the measured signal, this can help you produce more accurate measurements. Filters are not available for digitize functions.

The DAQ6510 filter options are repeating average, moving average, and hybrid average.

The repeating average filter produces slower results, but produces more stable results than the moving average filter. For all methods, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

The moving average filter adds measurements to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack. When the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements.

The hybrid average filter is only available when the buffer style is set to Full. It is similar to the moving average filter, except that it adds the number of measurements defined by the count to the stack before making the first averaged measurement. This ensures that the filter buffer is filled before returning the first measurement.

The repeating average filter is the only filter option available for use with channels.

If you create test algorithms and you are using the averaging filters, make sure the algorithms clear the filter memory stacks at appropriate times to avoid averaging an inappropriate set of measurements.

When the filter is turned on, the filter is applied before any relative offset, math, or limit operations. Once the relative offset is applied, the next filtered reading has the relative offset applied before it is reported to the instrument. This means that when you use relative offset, the next reading may not be zero.

For example, if the filter size is set to 10, ten internal measurements are stored. Once the tenth measurement is made, the display or remote interface updates and returns the average of the ten readings.

For additional information about the order in which math, filters, offsets, and limits are applied, see [Order of operations](#) (on page 4-91).

Repeating average filter

When the repeating average filter is selected, a set of measurements are made. These measurements are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed, and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack must be completely filled before an averaged sample can be produced, but it provides more stable results.

Moving average filter

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

When the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements.

For example, if the filter size is four, the first measurement is copied to all four stack locations. Therefore, $(\text{Reading1} + \text{Reading1} + \text{Reading1} + \text{Reading1})/4$. The display and remote interface update after the first reading. With each additional measurement, the average updates:

$$(\text{Reading2} + \text{Reading1} + \text{Reading1} + \text{Reading1})/4$$

$$(\text{Reading3} + \text{Reading2} + \text{Reading1} + \text{Reading1})/4$$

$$(\text{Reading4} + \text{Reading3} + \text{Reading2} + \text{Reading1})/4$$

Hybrid average filter

When the hybrid average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. After the number of measurements defined by the count, a new averaged sample is produced using the next measurement and the data that is in the stack.

The hybrid average filter is only available when the buffer style is set to Full. It is similar to the moving average filter, except that it adds the number of measurements defined by the count to the stack before making the first averaged measurement. This ensures that the filter buffer contains meaningful data before returning the first measurement.

Filter window

The filter window sets the window for the averaging filter that is used for measurements for the selected function.

The noise window allows a faster response time to large signal step changes. A reading that is outside the plus or minus noise window fills the filter stack immediately.

If the noise does not exceed the selected percentage of range, the reading is based on an average of reading conversions, which is the normal averaging filter. If the noise does exceed the selected percentage, the reading is a single reading conversion, and new averaging starts from this point.

Setting up the averaging filter

Using the front panel:

1. Press the **MENU** key.
2. Under Measure, select **Calculations**.
3. For Filter, select **On** to enable filtering.
4. Select **Settings**.
5. For the Filter Type, select **Moving**, **Hybrid**, or **Repeat**.
6. For the Filter Count, enter the number of measurements to be made for each averaged measurement sample.
7. For the Filter Window, select a value.
8. Select **OK**.
9. Select **HOME** to return to the home screen to view the measurements with the filter applied.

NOTE

Once the filter is set up, you can enable and disable the filter from the SETTINGS swipe screen. When filtering is enabled, the FILT indicator on the home screen is lit.

Using SCPI commands:

To set the averaging filters using SCPI commands, refer to the following command descriptions:

[\[:SENSe\[1\]\]:<function>:AVERage:COUNT](#) (on page 12-125)

[\[:SENSe\[1\]\]:<function>:AVERage\[:STATe\]](#) (on page 12-127)

[\[:SENSe\[1\]\]:<function>:AVERage:TCONtrol](#) (on page 12-129)

[\[:SENSe\[1\]\]:<function>:AVERage:WINDow](#) (on page 12-131)

Using TSP commands:

To set the averaging filters using TSP commands, refer to the following command descriptions:

[dmm.measure.filter.count](#) (on page 14-230)

[dmm.measure.filter.enable](#) (on page 14-232)

[dmm.measure.filter.type](#) (on page 14-234)

[dmm.measure.filter.window](#) (on page 14-236)

Limit testing and binning

The DAQ6510 can be set up for limit testing and binning. It can perform simple benchtop limit testing using the front panel or sophisticated limit and binning operations using the trigger model and digital I/O to control external component-handling devices.

Some typical forms of limit testing include:

- Simple pass-or-fail testing.
- Resistor grading: Inspect multiple limits until the first failure is received.
- Resistor sorting: Inspect multiple limits until the first pass is received.

For binning applications, you use limit testing to determine placement of tested parts. To set up the instrument to place the part in the correct bin, you do the following steps:

- Determine and record a bin number for later use.
- Output a digital bit pattern to physically place the tested device in a bin.
- If multiple tests are performed on the same part, determine when the part should be binned:
 - Bin the part as soon as it fails a test.
 - Bin the part after all parameters are measured; bin according to the first failure or a combination of failures.

Limit testing allows you to set high and low limit values. When the reading falls outside these limits, the instrument displays L1FAIL or L2FAIL. The low limit must be set to the low value and the high limit must be set to the high value to prevent an automatic limit-fail.

The limit values are stored in volatile memory.

Limits are tested after any selected filter, relative offset, and math functions have been applied to the measurement.

The DAQ6510 provides two binning trigger-model templates to assist with setup, one for grading and one for sorting. These trigger-model templates are only available if a communications accessory card is installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232. Refer to [Trigger-model templates](#) (on page 8-59).

Limit testing using the front-panel interface

You can do pass or fail limit testing through the front panel. When limit testing and a test fails, the limit (1 or 2) that failed is shown on the home screen.

NOTE

The low value must be less than the high value. If it is not, an automatic limit fail will occur.

Using the front panel:

1. Press the **MENU** key.
2. Under Measure, select **Calculations**.
3. Set Limit 1 or Limit 2 to **On**.
4. Select **Settings**.
5. The Auto Clear setting automatically clears the limit fail indicator when a new passing measurement is made. To turn this feature off, select **Off**.
6. Set the **Low Value**. If the measurement is below the Low Value, the limit failure indicator is displayed.
7. Set the **High Value**. If the measurement is above the High Value, the limit failure indicator is displayed.
8. The Audible setting determines if a beeper sounds when a measurement passes or fails. Set as needed.
9. Select **HOME**.
10. Make a measurement. L1PASS is displayed if the measurement is in the limits; L1FAIL is displayed if the measurement is not in the limits.

An example of using limit testing to check resistors is described in the following topic.

Front-panel limit test to check resistors

This example is set up to test a box of $100\ \Omega \pm 1$ percent and $100\ \Omega \pm 10$ percent resistors that you need to separate manually. You can change values as needed to adapt the test to your needs.

To set up the test:

1. Press the **FUNCTION** key.
2. Select **4W Res.**
3. Press the **MENU** key.
4. Under Measure, select **Calculations.**
5. Set Limit 1 and Limit 2 to **On.**
6. Select **Settings** for Limit 1.
7. Set the High Value to **110 Ω .**
8. Set the Low Value to **90 Ω .**
9. Select **OK.**
10. Select **Settings** for Limit 2.
11. Set the High Value to **101 Ω .**
12. Set the Low Value to **99 Ω .**
13. Select **OK.**

Run the test:

1. Press the **HOME** key.
2. Use 4-wire connections to connect the first resistor to the instrument.
3. Verify that the instrument is set to Continuous Measurement. If necessary, hold the **TRIGGER** key for 2 s and select **Continuous Measurement**.
4. Observe the measurements. If the resistor is inside the limits set for Limit 1, L1PASS is displayed. If the resistor is not within the limits, L1FAIL is displayed. If the resistor is in the limits set for Limit 2, L2PASS is displayed. If the resistor is not within the limits, L2FAIL is displayed. An example of a test that passed the L1 test but failed the L2 test is shown below.

Figure 86: Limit test pass and fail indicators

Line cycle synchronization

Using line synchronization helps increase common-mode and normal-mode noise rejection. When line cycle synchronization is enabled, measurements are initiated at the first positive-going zero crossing of the power line cycle after the trigger.

Line cycle synchronization only applies to the following functions: Voltage, current, temperature, continuity, resistance, and DC voltage ratio.

You can enable line synchronization for NPLC measurements, which increases the normal-mode rejection ratio (NMRR) and common-mode rejection ratio (CMRR).

Using aperture or NPLCs to adjust speed and accuracy

You can adjust the amount of time that the input signal is measured. Adjustments to the amount of time affect the usable measurement resolution, the amount of reading noise, and the reading rate of the instrument.

NOTE

This topic discusses aperture for the measure functions. For information regarding aperture for the digitize functions, refer to [Digitize functions](#) (on page 4-35).

Depending on the function, you can set the time as an aperture or number of power line cycles (NPLCs).

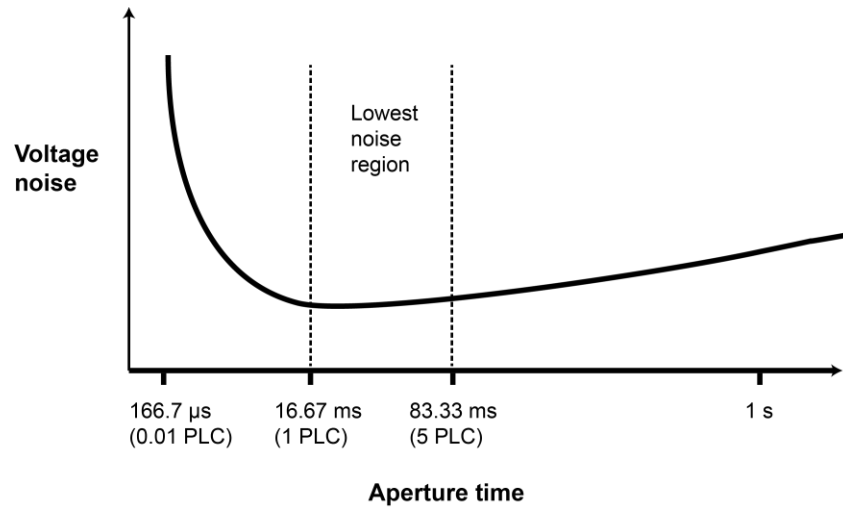
When you set the time as an aperture, you set it as a number of seconds.

When you set the time in relation to NPLCs, you set it as the number of power line cycles that should occur during the measurement. Each power line cycle for 60 Hz is 16.67 ms (1/60); for 50 Hz or 400 Hz, it is 20 ms (1/50).

The shortest amount of time or lowest NPLC value results in the fastest reading rate but increases the reading noise and decreases the number of usable digits.

The longest amount of time or highest NPLC value provides the lowest reading noise and more usable digits, but has the slowest reading rate.

The DAQ6510 has a nonlinear shape for its speed versus noise characteristics. The DAQ6510 is optimized for the 1 PLC to 5 PLC reading rate. At these rates (lowest noise region in graph), the DAQ6510 will make corrections for its own internal drift and will still be fast enough to settle a step response of less than 100 ms.

Figure 87: Speed compared to noise characteristics

When using NPLCs to adjust the rate, frequency and period cannot be set. However, when using aperture to adjust the rate, aperture can be set for both frequency and period.

NOTE

The DAQ6510 uses internal references to calculate an accurate and stable reading. When the NPLC setting is changed, each reference is automatically updated to the new NPLC setting before a reading is generated. Therefore, frequent NPLC setting changes can result in slower measurement speed.

This setting also affects the normal mode rejection ratio (NMRR) and common mode rejection ratio (CMRR). Normal mode noise is the noise signal between the HI and LO terminals; common-mode noise is the noise signal between LO and chassis ground. See the DAQ6510 specification for NMRR and CMRR values at different PLC settings.

If you change the aperture or NPLCs, you may want to adjust the displayed digits to reflect the change in usable digits. Refer to [Setting the number of displayed digits](#) (on page 3-75).

For functions that can accept either an aperture or an NPLC value, changing the value of one changes the value for the other. For example, if you set an aperture of 0.035, then set an NPLC value of 2, the aperture value is changed to 0.033333333.

To set NPLC using the front panel:

1. Press the **FUNCTION** key.
2. Select the measure function.
3. Press the **MENU** key.
4. Under Measure, select **Settings**.
5. Select **Integration Unit**. If the function allows both NPLC or aperture settings, the Integration Rate dialog box is displayed. Otherwise, a number pad is displayed.
6. If the Integration Unit dialog box is displayed, set the Unit to be **NPLC** or **Aperture**.
7. For NPLC or Aperture, enter the value.
8. Select **OK**.

DMM resistance measurement methods

The method that the DAQ6510 uses to measure resistance depends on the resistance range. For resistance ranges from 1 Ω to 1 M Ω , the DAQ6510 uses the constant-current method to measure resistance. For resistance ranges 10 M Ω and 100 M Ω , the ratiometric method is used.

When the constant-current method is used, the DAQ6510 sources a constant current (I) to the device under test and measures the voltage (V). Resistance (R) is then calculated and displayed using the known current and measured voltage ($R = V/I$).

When the ratiometric method is used, test current is generated by a 6.9 V reference through a 10 M Ω reference resistance (R_{REF}).

Constant-current source method

For the 1 Ω to 1 M Ω ranges, the DAQ6510 uses the constant-current method to measure resistance. The DAQ6510 sources a constant current (I_{SOUR}) to the device under test (DUT) and measures the voltage (V_{MEAS}). Resistance (R_{DUT}) is then calculated and displayed using the known current and measured voltage.

Simplified schematics of the constant-current method are shown in the following figures. The test current sourced to the DUT depends on the selected measurement range. For example, for the 100 Ω range, the test current is 1 mA. Because the voltmeter of the DAQ6510 has high input impedance (>10 G Ω), virtually all the test current (1 mA) flows through the DUT. For a DUT that is ≤ 1 k Ω , 4-wire ohms measurements should be used, as shown in the next figure. Because the voltage is measured at the DUT, voltage drop in the test leads is eliminated (this voltage could be significant when measuring a low-ohm DUT).

Figure 88: Two-wire constant-current source method

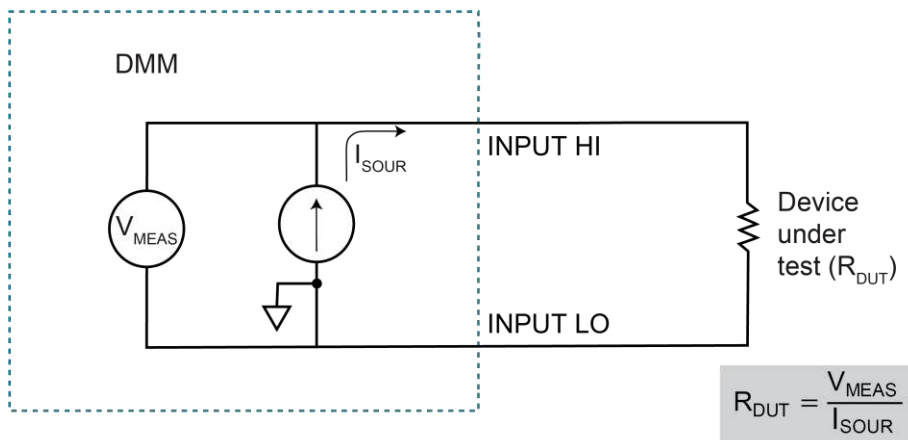
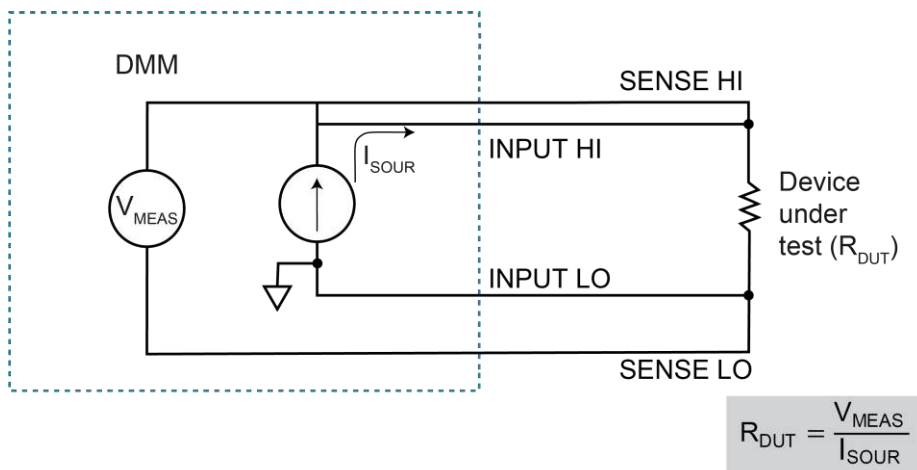
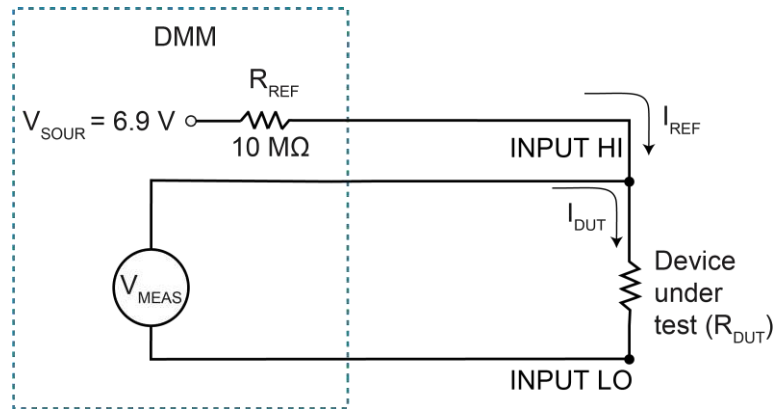


Figure 89: Four-wire constant-current source method

Ratiometric method

For the 10 M Ω and 100 M Ω ranges, the ratiometric method is used to measure resistance. Test current for this method is generated by a 6.9 V voltage source through a 10 M Ω reference resistance (R_{REF}), as shown in the figure below.

Basic circuit theory dictates that I_{REF} is equal to the I_{DUT} . Because the voltmeter of the DAQ6510 (V_{MEAS}) has high input impedance (>10 G Ω), current through the voltmeter branch is insignificant and can be discounted. Therefore, as shown in the following figures, $I_{REF} = I_{DUT}$.

Figure 90: 2-wire ratiometric resistance measurement method schematic

$$I_{REF} = I_{DUT}$$

$$\frac{V_{SOUR} - V_{MEAS}}{R_{REF}} = \frac{V_{MEAS}}{R_{DUT}}$$

$$R_{DUT} = \frac{V_{MEAS}}{V_{SOUR} - V_{MEAS}} \times R_{REF}$$

For R_{DUT} of approximately 0 ohms

$$I_{REF} = \frac{V_{SOUR}}{R_{REF}}$$

$$R_{DUT} \approx 10 \text{ ohms}$$

$$I_{REF} = \frac{V_{SOUR}}{R_{REF} + R_{DUT}}$$

$$I_{REF} = \frac{V_{SOUR}}{2R_{REF}}$$

Because $I = V/R$, the equation is modified using the V/R equivalents in place of I_{REF} and I_{DUT} . Therefore:

$$I_{SOUR} = (V_{MEAS} / R_{REF}) + (V_{MEAS} / R_{DUT})$$

NOTE

V_{MEAS} is measured by the DAQ6510. With V_{MEAS} , I_{SOUR} , and R_{REF} known, the DAQ6510 calculates the resistance of the DUT and displays the result. R_{REF} is learned during calibration and V_{SOUR} is routinely self-calibrated when autozero is enabled.

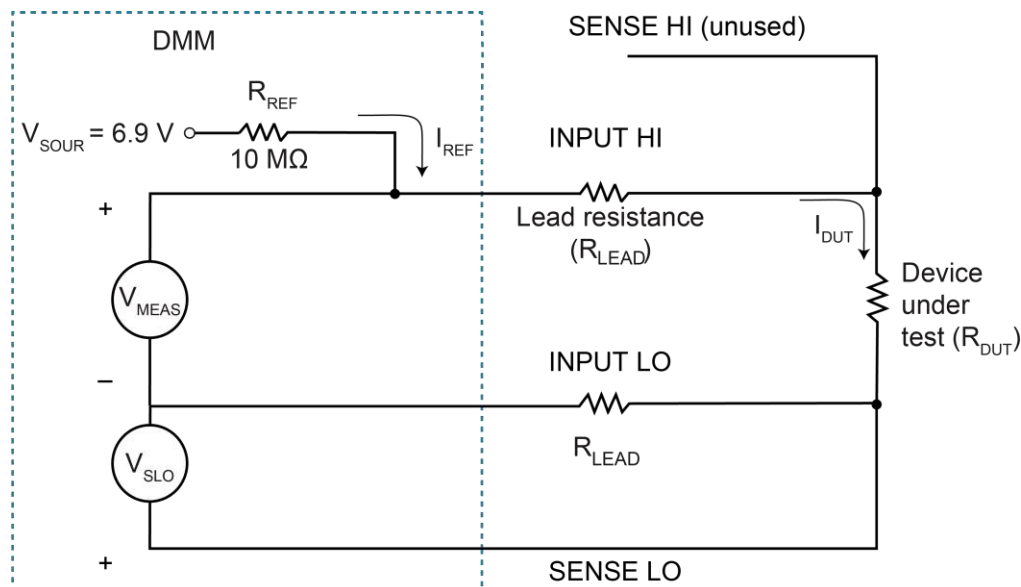
As shown, the 4-wire ohm function can also be used to measure ohms for the 10 M Ω and 100 M Ω ranges. To minimize the effects of charge injection when autozero is enabled, the measurements of the 10 M Ω to 100 M Ω ranges are actually 3-wire ohm measurements. SENSE HI is not used (it can be left open). The measurement method is similar to the ratiometric method for 2-wire ohms, but it performs an extra voltage measurement (V_{LEAD}) to compensate for voltage drop in the input test leads.

NOTE

V_{MEAS} includes the voltage drops of the input test leads (Input HI and Input LO). Therefore, the actual voltage drop across the DUT is V_{MEAS} minus the two voltage drops in the test leads. Because matched inputs are used, the voltage drop is $2 \times V_{\text{LEAD}}$. Therefore:

$$V_{\text{DUT}} = V_{\text{MEAS}} - 2(V_{\text{LEAD}})$$

Figure 91: 4-wire ratiometric resistance measurement method



Low-level voltage measurement considerations

Low-level voltage measurements can be adversely affected by noise or other unwanted signals that can make it difficult to get accurate voltage readings. Some of the phenomena that can cause unwanted noise include thermoelectric effects (thermocouple action), source resistance noise, magnetic fields, and radio frequency interference. The following paragraphs discuss the most important of these effects and ways to minimize them.

NOTE

For comprehensive information on low-level measurements, see the *Low Level Measurements Handbook*, which is available from tek.com/keithley (tek.com/products/keithley).

Thermoelectric potentials

Thermoelectric potentials, or thermoelectric EMFs, are the most common source of errors in low-voltage measurements. These small electric potentials are generated when different parts of the circuit are at different temperatures and when conductors made of dissimilar metals are joined.

Thermoelectric EMFs can cause the following conditions:

- Instability or zero offset is much higher than expected.
- The reading is sensitive to and responds to temperature changes. This effect can be demonstrated by placing a heat source near the circuit or by observing changes because of sunlight or the activation of heating and air conditioning systems.

The following paragraphs discuss how thermoelectric potentials are generated and ways to minimize their effects.

Thermoelectric coefficients

The following table shows the magnitude of thermoelectric EMFs that are generated for different materials. Best results are obtained with clean copper-to-copper connections.

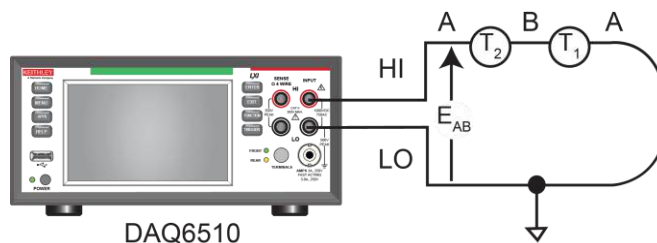
Material thermoelectric coefficients	
Material	Thermoelectric potential
Copper-to-copper	0.2 $\mu\text{V}/^\circ\text{C}$
Copper-to-silver	0.3 $\mu\text{V}/^\circ\text{C}$
Copper-to-gold	0.3 $\mu\text{V}/^\circ\text{C}$
Copper-to-cadmium/tin	0.3 $\mu\text{V}/^\circ\text{C}$
Copper-to-lead/tin	1 $\mu\text{V}/^\circ\text{C}$ to 3 $\mu\text{V}/^\circ\text{C}$
Copper-to-Kovar [®]	40 $\mu\text{V}/^\circ\text{C}$ to 75 $\mu\text{V}/^\circ\text{C}$
Copper-to-silicon	400 $\mu\text{V}/^\circ\text{C}$
Copper-to-copper oxide	1000 $\mu\text{V}/^\circ\text{C}$

Thermoelectric EMF generation

The figure below shows how thermoelectric EMFs are generated.

The test leads are made of material A, while the source under test is material B. The temperatures between the junctions are shown as T_1 and T_2 .

Figure 92: Thermoelectric EMF generation



To calculate the thermoelectric EMFs that are generated:

$$E_{AB} = Q_{AB} (T_1 - T_2)$$

Where:

- E_{AB} is the generated thermoelectric EMF
- Q_{AB} is the thermoelectric coefficient of material A with respect to material B ($\mu\text{V}/^\circ\text{C}$)
- T_1 is the temperature of the B junction ($^\circ\text{C}$ or K)
- T_2 is the temperature of the A junction ($^\circ\text{C}$ or K)

A typical test setup has several copper-to-copper junctions. Each junction can have a thermoelectric coefficient as high as $0.2 \mu\text{V}/^\circ\text{C}$. Since the two materials frequently have several degrees of temperature differential, thermoelectric EMFs of several microvolts can be generated even if reasonable precautions are taken.

Minimizing thermoelectric EMFs

To minimize thermoelectric EMF generation:

- Construct circuits that use the same material for all conductors. For example, connections made by crimping copper sleeves or lugs on copper wires result in copper-to-copper junctions, which generate minimal thermoelectric EMFs.
- Keep connections clean and free of oxides.
- Use low-thermoelectric cables and connections.
- Keep the two materials forming the junction at the same temperature.
- Keep the two junctions close together.
- Allow test equipment to warm up and reach thermal equilibrium in a constant ambient temperature.
- Keep all junctions away from air currents; in some cases, it may be necessary to thermally insulate sensitive junctions to minimize temperature variations.
- When making a copper-to-copper connection, apply sufficient pressure to ensure the connection is gastight to prevent future oxidation.
- In some cases, you may need to connect the two thermal junctions together with good thermal contact to a common heat sink. Unfortunately, most good electrical insulators are poor heat conductors. In cases where low thermal conductivity may be a problem, you can use special insulators that combine high electrical insulating properties with high thermal conductivity. Some examples of these materials include hard anodized aluminum, sapphire, and diamond.

Using relative offset to minimize thermoelectric EMFs

Some systems may still have residual thermoelectric offsets after following the guidelines in [Minimizing thermoelectric EMFs](#) (on page 4-82). If the offsets are relatively constant, you can use the relative offset feature in the DAQ6510 to cancel them. Refer to [Relative offset](#) (on page 4-56) for information.

Magnetic fields

When a conductor loop cuts through magnetic lines of force, a very small current is generated. This phenomenon can cause unwanted signals to occur in the test leads of a test system. If the conductor has sufficient length or cross-sectional area, even weak magnetic fields can create signals that affect low-level measurements.

To reduce these effects:

- Reduce the lengths of the connecting cables.
- Minimize the exposed circuit area.
- Change the orientation of the leads or cables.
- Minimize cable loop area or introduce cable twisting.

In extreme cases, you may require magnetic shielding. Special metal with high permeability at low flux densities (such as mu metal) is effective at reducing these effects.

Even when the conductor is stationary, you may have problems with magnetically-induced signals. Fields can be produced by sources such as the AC power line voltage and large inductors, such as power transformers. Keep the DAQ6510 voltage source and connecting cables away from these potential noise sources.

Radio frequency interference

Radio frequency interference (RFI) is a general term used to describe electromagnetic interference over a wide range of frequencies across the spectrum. RFI creates problems at low signal levels, but it can also affect measurements at high levels if the fields are of sufficient magnitude.

RFI can be caused by steady-state sources, such as radio or TV signals, or some types of electronic equipment, such as microprocessors and high-speed digital circuits. It can also result from impulse sources, as in the case of arcing in high-voltage environments. The effect on the measurement can be considerable if enough of the unwanted signal is present.

You can minimize RFI in several ways:

- Keep the DAQ6510 voltage source and signal leads away from RFI sources.
- Shield the instrument, signal leads, sources, and other measuring instruments.
- In extreme cases, a specially constructed screen room may be required to sufficiently attenuate the RFI signal.

In some situations, the DAQ6510 digital filter may help to reduce RFI effects, but additional external filtering may be required. Filtering may have detrimental effects, such as increased settling time on the signal.

Shielding

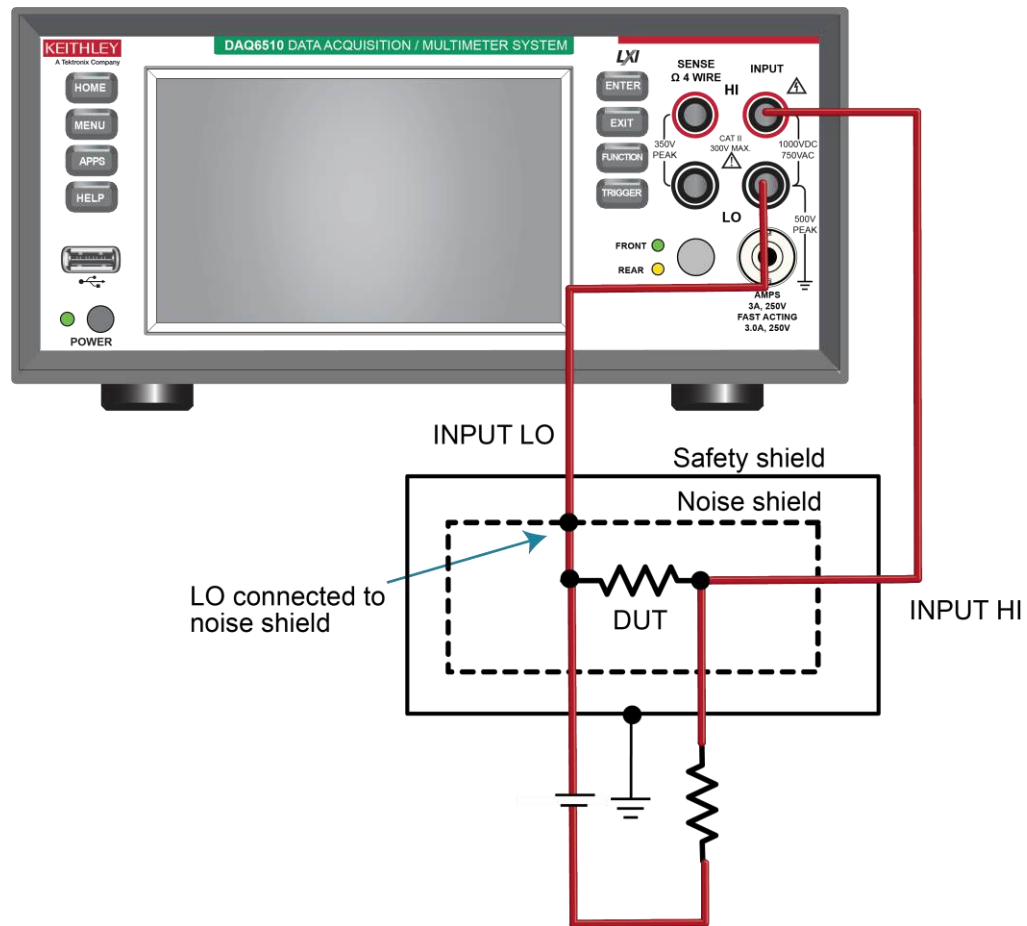
AC voltages that are extremely large compared with the DC signal to be measured may produce an erroneous output. To minimize AC interference, the circuit should be shielded, with the shield connected to the DAQ6510 input low (particularly for low-level sources). Improper shielding can cause the DAQ6510 to behave in one or more of the following ways:

- Unexpected offset voltages
- Inconsistent readings between ranges
- Sudden shifts in readings
- Higher overall noise in the measurements

To minimize pick-up, keep the voltage source and the DAQ6510 away from strong AC magnetic sources. The voltage induced due to magnetic flux is proportional to the area of the loop formed by the input leads. Minimize the loop area of the input leads and connect each signal at only one point. You may also need to provide shielding if you have high voltage, high frequency input signals. These types of signal can cause problems with the DAQ6510 display.

To minimize noise, you may need a closed metal shield that surrounds the source. This shield should be connected to input LO in most cases. In some situations, you may get better noise performance with the shield connected to chassis ground.

Connect the safety shield to a known safety earth ground using #18 AWG or higher wire.

Figure 93: DAQ6510 front-panel noise shield connections

⚠ WARNING

INPUT and SENSE LO are not internally connected to the chassis and cannot be allowed to float above chassis ground more than the values shown on the front panel. Failure to follow this guideline can result in injury, death, or instrument damage.

Measurement settling considerations

If you apply high-power signals (more than 300 V_{RMS}, 500 VDC, 1 A DC or 1 A_{RMS}), the signal-conditioning components may self-heat. These errors are included in the instrument specifications. Internal temperature changes due to self-heating may cause additional errors on other functions or ranges. The additional error normally dissipates in a few minutes.

On the AC Voltage and Frequency functions, if you attempt to measure an input following a DC offset voltage change, errors occur. The input-locking resistor-capacitor (RC) time constant must be allowed to fully settle (up to 3 seconds) before the most accurate measurements are possible.

Reading settling times are also affected by the source impedance, cable dielectric characteristics, and thermal EMF of connections. Keithley recommends the use of polytetrafluoroethylene (PTFE) or other high-impedance, low-dielectric absorption wire insulation for these measurements. To maintain low thermal EMF, connectors and wires made of copper are recommended.

Reference junctions

A reference junction is the cold junction in a thermocouple circuit that is held at a stable, known temperature. The cold junction is where dissimilar wire connections must be made. As long as the temperature of the cold junction is known, the DAQ6510 can factor in the reference temperature to calculate the actual temperature reading at the thermocouple.

The standard reference temperature is the ice point (0 °C). The ice point can be precisely controlled, and the National Institute of Standards and Technology (NIST) uses it as the fundamental reference for its voltage-to-temperature conversion tables. However, other known temperatures can be used.

The internal reference junction is only valid on the rear inputs for CJC scan cards. If used on the front or without a CJC scan card, the resulting data is inaccurate.

The DAQ6510 can acquire the cold junction temperature by measuring the cold junction using a thermistor or 4-wire RTD, or you can enter a known temperature value.

The reference junction types supported by the DAQ6510 are:

- Simulated reference junction
- Internal reference junction
- External reference junction

These reference junctions are explained in the following paragraphs.

Simulated reference junction

The simulated reference temperature for the DAQ6510 can be set from 0 °C to 65 °C. The DAQ6510 measures the input voltage and factors in the simulated reference temperature to calculate the temperature reading at the thermocouple.

An example of a simulated reference junction is an ice bath. The input wire to thermocouple wire connections are immersed (but electrically isolated) in the ice bath, and the user enters the 0 °C simulated reference temperature into the DAQ6510.

NOTE

The most accurate temperature measurements are achieved by using a simulated reference junction using an ice-point reference.

Internal reference junction

When the internal reference junction is selected, the DAQ6510 measures the temperature of the cold junction (screw terminals), measures the input voltage, and then calculates the temperature reading at the thermocouple. For specific switching modules, the cold junction can be the switching module's screw terminals with voltage temperature sensors strategically placed to measure the temperature of the cold junction.

To help maintain stability and accuracy over time and changes in temperature, the DAQ6510 periodically measures internal voltages corresponding to offsets (zero) and amplifier gains. For thermocouple temperature measurements using the internal reference junction, the internal temperature is also measured. These measurements are used in the algorithm to calculate the reading of the input signal. This process is known as autozeroing.

The internal reference junction is only valid on the rear inputs for cards that support CJC, such as the 7700, 7706, 7708, and 7710 switching modules.

NOTE

Internal temperature references are collected whether or not autozero is enabled.

External reference junction

Thermocouple readings using the rear terminals can be configured to use an external reference junction setting. The DAQ6510 assumes the external reference junction is connected to channel 1 of a slot. It is recommended that this channel be configured for thermistor or RTD temperature measurements. However, the instrument does not verify the configuration. Each time a reading is made on the external reference junction channel, it is used as the new external reference junction value in subsequent external reference readings. External reference readings work with `channel.close` and scanning.

For nonsimulated thermocouple measurements, first make a thermistor or RTD measurement before enabling the external reference junction.

The following code shows how to set up an external reference junction using TSP commands.

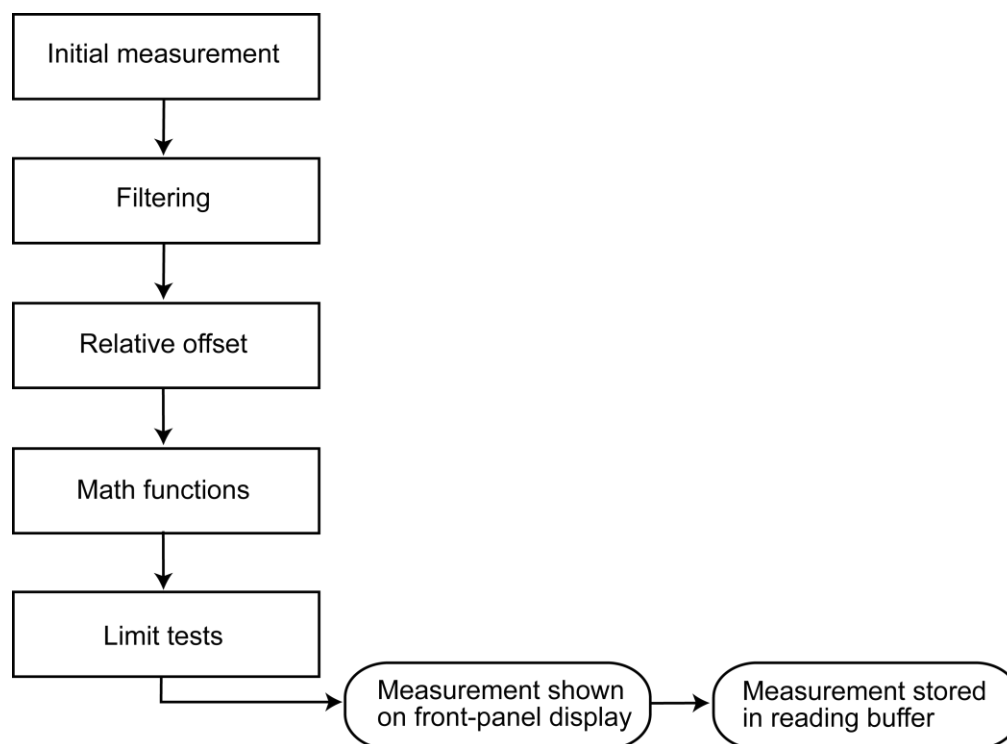
```
reset()
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101", dmm.ATTR_MEAS_TRANSDUCER, dmm.TRANS_THREERTD)
channel.setdmm("101", dmm.ATTR_MEAS_THREE_RTD, dmm.RTD_D100,
    dmm.ATTR_MEAS_SIM_REF_TEMP, 30)
channel.setdmm("101", dmm.ATTR_MEAS_OPEN_DETECTOR, dmm.ON,
    dmm.ATTR_MEAS_OFFCOMP_ENABLE, dmm.ON)
channel.setdmm("101", dmm.ATTR_MEAS_REF_JUNCTION, dmm.REFJUNCT_EXTERNAL)
scan.measurecount = 1
extRefJunc = buffer.make(20)
buffer.clearstats()
buffer.saveappend(extRefJunc, "/usb1/MyData.csv")
channel.close("101")
dmm.measure()
channel.close("102")
for i = 1, 4 do
    print(dmm.measure())
end
channel.open("allslots")
```

Order of operations

The measurements have filtering, relative offset values, math operations, and limit testing applied to them in a predetermined order. The measurements that are displayed on the front panel and stored in the reading buffers represent the measurements with any selected operations applied to them.

These operations are applied to the measurement as shown in the following figure.

Figure 94: DAQ6510 order of operations



For more information on these operations, see the following topics:

- [Filtering measurement data](#) (on page 4-65)
- [Relative offset](#) (on page 4-56)
- [Calculations that you can apply to measurements](#) (on page 4-59)
- [Limit testing and binning](#) (on page 4-69)

Saving setups

You can save the present settings, scan settings, watch channels, and any configuration lists that you have defined for the DAQ6510 to internal memory or an external USB flash drive. If a channel is closed, the state is recorded in the saved setup, but is not implemented when you recall the setup. All channels are open when you recall the setup.

After the settings are saved, you can recall the settings. You can also set them to be the default settings when the instrument is powered on.

If you are using TSP commands, saved setups are scripts and can be added, modified, and deleted like any other script. See [Introduction to TSP operation](#) (on page 13-1) for additional information about working with scripts.

NOTE

Settings made on the Graph and Histogram tabs are not saved as part of a saved setup. To record graph settings, you can press HOME and ENTER to save an image of the settings with the screen capture feature. Refer to [Save screen captures to a USB flash drive](#) (on page 3-79) for additional information.

Save a user setup to internal memory

From the front panel:

1. Configure the DAQ6510 to the settings that you want to save.
2. Press the **MENU** key.
3. Under Scripts, select **Save Setup**.
4. Select **Create**. A keyboard is displayed.
5. Use the keyboard to enter the name of the script.
6. Select the **OK** button on the displayed keyboard. The script is added to internal memory.

Using SCPI commands:

1. Configure the instrument to the settings that you want to save.
2. Send the command:
`*SAV <n>`
Where <n> is an integer from 0 to 4.

NOTE

In the front-panel script menus, the setups saved with the `*SAV` command have the name `Setup0x`, where `x` is the value you set for <n>.

Using TSP commands:

1. Configure the instrument to the settings that you want to save.
2. Send the command:
`createconfigscript("setupName")`
Where `setupName` is the name of the setup script that is created.

Save a user setup to a USB flash drive

NOTE

You cannot save to the flash drive using SCPI commands.

From the front panel:

1. Save the user setup to internal memory, as described in [Save a user setup to internal memory](#) (on page 4-93).
2. Insert the USB flash drive into the USB port on the front panel.
3. Press the **MENU** key.
4. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
5. In the Internal Scripts list, select the script you want to copy to the USB flash drive.
6. Select **>**. The file is transferred to the USB flash drive, and the corresponding file name is displayed in the USB Scripts box.

Using TSP commands:

1. Save the user setup to internal memory, as described in [Save a user setup to internal memory](#) (on page 4-93).
2. Insert the USB flash drive into the USB port on the front panel.
3. Send the command:

```
setupName.save("/usb1/USBSetupName")
```

Where *setupName* is the name of the user setup and *USBSetupName* is the name of the file on the USB flash drive. You can use the same name for *setupName* and *USBSetupName*.

Copy a user setup

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

1. Insert the USB flash drive into the USB port on the front panel.
2. Press the **MENU** key.
3. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
4. In the USB Scripts list, select the script you want to copy from the USB flash drive.
5. Select **<**. The file is transferred to the instrument, and the corresponding file name is displayed in the Internal Scripts box.

Delete a user setup

To remove a user setup from internal memory or the USB flash drive from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
3. Under Internal Scripts or USB Scripts, select the name of the script.
4. Select **Delete**. A confirmation message is displayed.
5. Select **OK**.

To delete a user setup from internal memory using SCPI commands:

You must overwrite an existing setup with the new setup. See [Save a user setup to internal memory](#) (on page 4-93).

To delete a user setup from internal memory using TSP commands, send the command:

```
script.delete("setupName")
```

Where *setupName* is the name of the script that will be deleted.

Recall a user setup

You can recall setups from internal nonvolatile memory or a USB flash drive. When you recall a setup, you run a script that restores the instrument to the settings that are saved in that script.

To recall a saved setup from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Run**.
3. In the Available Scripts list, select the script you want to recall. USB scripts have the prefix `usb1/`.
4. Select **Run Selected**.

To recall a user setup from internal memory using SCPI commands, send the command:

```
*RCL <n>
```

Where *<n>* is an integer from 0 to 4 that represents the saved script.

To recall a saved setup using TSP commands, send the command:

```
setupName()
```

Where *setupName* is the name of the script that contains the setup that was saved with `createconfigscript()`.

Define the setup used when power is turned on

You can select a configuration to be used when power is turned on.

From the front panel:

1. Set the instrument to the settings that you want it to have each time the power is turned on.
2. Press the **MENU** key to open the main menu. Under Scripts, select **Save Setup**.
3. Select **Create**. A keyboard is displayed.
4. Enter the name of the new script, and then select **ENTER** on the keyboard to save it. The instrument saves all present system settings to the script and displays a confirmation message.
5. Select **OK**.
6. Press the **EXIT** key to return to the main menu.
7. Under Scripts, select **Run**. The RUN SCRIPTS window opens.
8. Select the script you created.
9. Select **Copy to Power Up**.
10. Select **OK** on the confirmation message.

Using a SCPI command, send the command:

```
:SYSTem:POSetup <name>
```

Where <name> is:

- RST: Use the *RST defaults.
- SAV0: Use the setup stored at memory location 0
- SAV1: Use the setup stored at memory location 1
- SAV2: Use the setup stored at memory location 2
- SAV3: Use the setup stored at memory location 3
- SAV4: Use the setup stored at memory location 4

Using a TSP command:

Save the script that you want to use as the power-on default to be `autoexec`. For example, to save the commands that are presently in the instrument to be the power-on defaults, send the command:

```
createconfigscript("autoexec")
```

NOTE

If an `autoexec` script already exists, you must delete it by sending the `script.delete("autoexec")` command. Performing a system reset does not delete the `autoexec` script.

Saving front-panel settings into a macro script

You can save some settings made through the front panel into a macro script that you can run later.

The settings that are saved include any settings made through:

- Measure menu Settings, Calculations, Reading Buffers, scanning, and QuickSet (except the Performance slider, which cannot be used when recording a macro)
- Trigger menu options Templates and Configure
- The Graph Trigger tab
- System Communication
- Time and date

NOTE

Only settings are stored; no front-panel only options or key presses are stored.

It also saves the reading format, interface access, and system reset settings.

Macro scripts are limited to 10 kB per script.

Recording a macro script

To record a macro script:

1. Press the **MENU** key.
2. Under Scripts, select **Record**.
3. Select the **Start Macro** button.
4. Make the settings that you want to record.
5. Press the **MENU** key.
6. Under Scripts, select **Record**.
7. Select the **Stop Macro** button. The Macro Script Name dialog box is displayed.
8. Enter a name for the script.
9. Select the **OK** button.

NOTE

You can also stop or cancel recording from the home screen. Select the **Recording** indicator in the indicator bar.

After you create a macro script, you can use the other Scripts menu options to run and manage scripts. Refer to [Scripts menu](#) (on page 3-63) for information on the options.

Running a macro script

You can run a macro script from the front panel or from a remote interface.

To run a macro script from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Run**.
3. Select the macro script to run.
4. Select **Run Selected**.

Using SCPI commands:

```
SCript:RUN "scriptName"
```

Where *scriptName* is the name of the macro script to run.

Using TSP commands:

```
scriptVar.run()
```

Where *scriptVar* is the name of the macro script to run.

Front-panel macro recording limitations

When you are recording a macro script from the front panel, the settings you make are recorded at the speed at which you make them. However, when the macro you created is run, it runs at remote command processing speed. This can be a problem when working with trigger models and other features that require time to finish processing before remaining commands can process.

For example, if you record a macro that includes a trigger model that you initiate followed by other settings changes or additional trigger initiate actions, an error message is generated. This is because the trigger model takes time to complete, but the macro recording from the front panel does not add `waitcomplete()` commands or other delay settings to the script that allow the trigger model to finish before processing the other commands.

Configuration lists

A configuration list is a list of stored settings for the measure or digitize function. You can restore these settings to change the function and its settings that are used by the instrument.

Configuration lists allow you to store the function settings of the instrument and then return the instrument to those settings as needed.

The instrument also uses configuration lists internally to manage the settings for scans.

You can recall configuration lists from the front panel, using remote commands, or as part of a trigger model or scan.

NOTE

Do not change the configuration lists that are created by the instrument directly. Use the front-panel options or remote commands to make changes to scans.

Configuration lists contain the function setting and the settings for the function, such as the NPLC, display digits, and math settings.

Scan and channel settings are not stored in configuration lists.

Configuration indexes

A configuration index contains a copy of all instrument measure settings at a specific point. Configuration lists are typically made up of multiple indexes.

You can store a maximum of 300,000 indexes.

The first time you store a configuration index, the instrument stores the settings in configuration index 1. Subsequent indexes are numbered sequentially. You can use the index number to identify a specific configuration index and perform operations on it, such as when using the ConfigList trigger-model template.

The settings that are stored in configuration list indexes are listed in [Settings stored in a measure configuration index](#) (on page 4-111) and [Digitize settings stored in a measure configuration list index](#) (on page 4-114).

Working with configuration lists and indexes

To create a configuration index, you need to:

- Create the configuration list
- Configure the instrument with the settings that you want to store in a configuration index
- Store the settings into a configuration index in the specified configuration list

After you store configuration indexes to a configuration list, you can do the following operations:

- Recall a configuration index and restore the instrument to the stored settings
- View the contents of a configuration index
- Delete a configuration index
- Delete the entire configuration list

You can work with configuration lists from the front panel or by using remote commands.

Create a configuration list

This example creates a configuration list named `MyMeasList`.

To use the front panel to create a measure configuration list:

1. Press the **MENU** key.
2. Under Measure, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select **Create New**. If a list already exists, choose **Select** and choose **Create New**. The keypad is displayed.
4. Enter a name for the configuration list you are creating. For this example, enter `MyMeasList`.
5. Select the **OK** button on the displayed keyboard. The MEASURE CONFIGURATION LISTS screen is displayed.

To use SCPI commands to create a measure configuration list:

```
:SENSe:CONFIguration:LIST:CREate "MyMeasList"
```

To use TSP commands to create a measure configuration list:

```
dmm.measure.configlist.create("MyMeasList")
```

Store settings into a configuration list index

This section describes how to store instrument settings to an index in a configuration list.

A configuration index contains a copy of the instrument or measure settings for a function at a specific time. You can store up to 300,000 indexes.

The following examples make settings on the instrument and stores them in configuration list `MyMeasList`.

Store settings using the front panel

To configure the instrument and store the settings into indexes:

1. Press the **FUNCTION** key.
2. Select **DC Current**.
3. Press the **MENU** key.
4. Under Measure, select **Settings**. The MEASURE SETTINGS menu is displayed.
5. Set the Range to **10 mA**.
6. Set NPLC to **1.00**.
7. Press the **MENU** key.
8. Under Measure, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
9. Select **MyMeasList**.
10. Select **Add Settings**. The configuration index is displayed on the list.
11. Press the **MENU** key.
12. Under Measure, select **Settings**. The MEASURE SETTINGS menu is displayed.
13. Change NPLC to **2**.
14. Press the **MENU** key.
15. Under Measure, select **Config Lists**.
16. Select **Add Settings**. The configuration index is displayed on the list with the differences from the first index. If there are no differences, "No change" is displayed for that index.

Store settings using SCPI commands

This example:

- Creates the measure configuration list `MyMeasList`.
- Sets the measure function to DC voltage.
- Sets the measure range to 100 V.
- Stores the settings to the configuration list `MyMeasList`.
- Sets the measure function to DC current.
- Sets the measure range to 100 mA.
- Stores the settings to the configuration list `MyMeasList`.

Send the following SCPI commands:

```
:SENSe:CONFIguration:LIST:CREate "MyMeasList"  
:FUNction "VOLTage"  
:SENSe:VOLTage:RANGe 100  
:SENSe:CONFIguration:LIST:STORe "MyMeasList"  
:FUNction "CURRent"  
:SENSe:CURRent:RANGe 0.1  
:SENSe:CONFIguration:LIST:STORe "MyMeasList"
```

Store settings for the active function using TSP commands

This example:

- Creates the measure configuration list `MyMeasList`.
- Sets the measure function to DC voltage.
- Sets the measure range to 100 V.
- Stores the settings to the configuration list `MyMeasList`.
- Sets the measure function to DC current.
- Sets the measure range to 100 mA.
- Stores the settings to the configuration list `MyMeasList`.

Send the following TSP commands:

```
dmm.measure.configlist.create("MyMeasList")  
dmm.measure.func = dmm.FUNC_DC_VOLTAGE  
dmm.measure.range = 100  
dmm.measure.configlist.store("MyMeasList")  
dmm.measure.func = dmm.FUNC_DC_CURRENT  
dmm.measure.range = 0.1  
dmm.measure.configlist.store("MyMeasList")
```

Store settings for a function that is not active using TSP commands

You can set up a function that is not active and store it in a configuration list using the `dmm.measure.configlist.storefunc()` command and the `dmm.measure.setattribute` command. You can retrieve the settings for a function using the `dmm.measure.getattribute` command.

The store function command stores the settings for a specific function into a configuration list. The configuration list must be created before you use the store function command. Refer to [dmm.measure.configlist.storefunc\(\)](#) (on page 14-220) for detail on using the command.

The set attribute command sets a single attribute for the specified function. For details of the command and listings of the parameters that can be set, refer to [dmm.measure.setattribute\(\)](#) (on page 14-299).

The following example demonstrates how to use the store function and set attribute commands. This example:

- Creates the measure configuration list `MyMeasList`.
- Sets the measure range to 100 V for the DC voltage function.
- Stores the settings to the configuration list `MyMeasList`.
- Sets the measure range to 100 mA for the DC current function
- Stores the settings to the configuration list `MyMeasList`.

Send the following TSP commands:

```
dmm.measure.configlist.create("MyMeasList")
dmm.measure.setattribute(dmm.FUNC_DC_VOLTAGE, dmm.ATTR_MEAS_RANGE, 100)
dmm.measure.configlist.storefunc("MyMeasList", dmm.FUNC_DC_VOLTAGE)
dmm.measure.setattribute(dmm.FUNC_DC_CURRENT, dmm.ATTR_MEAS_RANGE, 0.1)
dmm.measure.configlist.storefunc("MyMeasList", dmm.FUNC_DC_CURRENT)
```

Recall a configuration index

You can recall the settings stored in a specific configuration index in a configuration list.

This example recalls configuration index 2 from `MyMeasList`.

Using the front panel to recall a configuration index:

1. Press the **MENU** key.
2. Under Measure, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Choose **Select**. A menu of available configuration lists is displayed.
4. Select **MyMeasList**. The configuration indexes in the list display.
5. Select the second configuration index.
6. Select **Recall Index**.

Using SCPI commands:

To recall index 2 from a measure configuration list:

```
:SENSe:CONFIguration:LIST:RECall "MyMeasList", 2
```

Using TSP commands:

To recall index 2 from a measure configuration list:

```
dmm.measure.configlist.recall("MyMeasList", 2)
```

View configuration list contents

You can display or print the contents of a specific configuration index. The contents that are returned include all the active settings that the instrument saved when you stored the configuration index.

The following examples demonstrate how to view configuration index 2 from `MyMeasList`.

View configuration list contents using the front panel

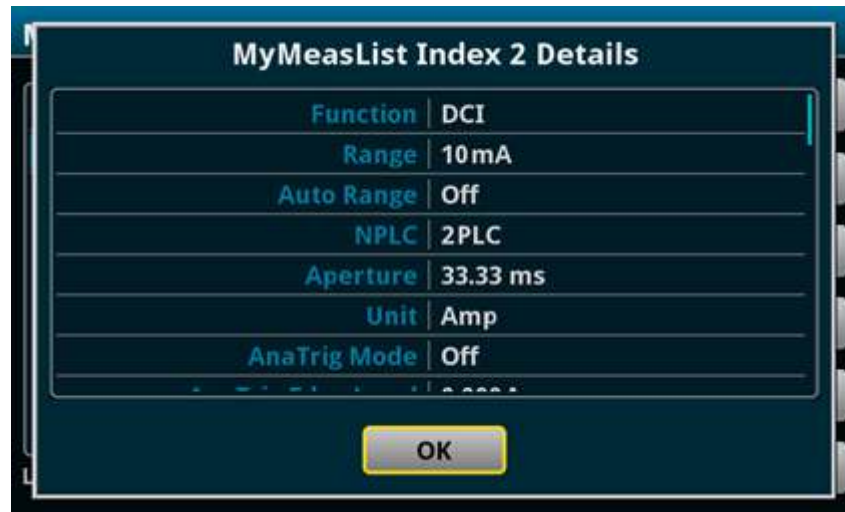
Use the following procedure to view configuration index 2 from `MyMeasList`.

You can use the Jump to Index option to go to an index in the list.

Use the front panel to view the contents of a measure configuration list:

1. Press the **MENU** key.
2. Under **Measure**, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select **MyMeasList**. The configuration indexes are displayed.
4. Select the second configuration index.
5. Select **Index Details**. The stored settings are displayed. You can scroll in the dialog box to view additional settings.

Figure 95: Measure configuration list index



6. When you are finished, select **OK**.

Using SCPI commands:

The SCPI configuration list query command returns a list of TSP commands that could be used to set the parameters stored in the specified configuration index.

To view a list of commands in configuration index 2 in a measure configuration list named `MyMeasList`, send the command:

```
:SENSe:CONFIguration:LIST:QUERy? "MyMeasList", 2
```

Using TSP commands:

The TSP configuration list query commands return a list of TSP commands that set the settings stored in the specified configuration index.

To print a list of commands in configuration index 2 in a measure configuration list named **MyMeasList**, send the command:

```
print(dmm.measure.configlist.query("MyMeasList", 2))
```

Delete a configuration index or list

You can delete individual configuration indexes from a configuration list using the front panel or remote commands.

When a configuration list index is deleted, the following indexes are renumbered so that the indexes are numbered sequentially.

NOTE

You cannot delete configuration lists from the front panel.

Use the front panel to delete a measure configuration index:

1. Press the **MENU** key.
2. Under Measure, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select **MyMeasList**. The configuration indexes are displayed.
4. Select the index.
5. Select **Remove Index**.

To use SCPI commands to delete index 2 from a measure configuration list named MyMeasList:

```
:SENSe:CONFIguration:LIST:DELeTe "MyMeasList", 2
```

To use SCPI commands to delete a measure configuration list named MyMeasList:

```
:SENSe:CONFIguration:LIST:DELeTe "MyMeasList"
```

To use TSP commands to delete index 2 from a measure configuration list named MyMeasList:

```
dmm.measure.configlist.delete("MyMeasList", 2)
```

To use TSP commands to delete a measure configuration list named MyMeasList:

```
dmm.measure.configlist.delete("MyMeasList")
```

Overwrite an existing index

When you are using the front panel, you can overwrite an existing index by selecting the index before selecting **Add Settings**. You are prompted to either overwrite the existing index or append the index to the end of the list.

When you are using remote commands, you can overwrite an existing index by specifying the existing index number in the index parameter of the command.

List the available configuration lists

You can view the names of the configuration lists stored on the instrument.

From the front panel:

On the MEASURE CONFIGURATION LIST screen, choose **Select** to display the list of configuration lists.

Using SCPI commands:

To receive the name of one measure configuration list stored on the instrument, use the following command.

```
:SENSe:CONFIguration:LIST:CATalog?
```

Each time this command executes, the name of one defined configuration is returned. To get all defined configuration lists, send this command until it returns an empty string. After the command returns an empty string, it wraps around and starts returning names again. If only an empty string is returned, no configuration lists exist.

Using TSP commands:

To receive the name of one measure configuration list stored on the instrument, use the following command.

```
print(dmm.measure.configlist.catalog())
```

Each time this command executes, the name of one defined configuration is returned. To get all defined configuration lists, send this command until it returns `nil`. After the command returns `nil`, it wraps around and starts returning names again. If only `nil` is returned, no configuration lists exist.

Determine the number of indexes in a configuration list

You can view the number of configuration indexes that are in a configuration list.

Use the front panel to view the number of indexes in a measure configuration list:

1. Press the **MENU** key.
2. Under **Measure**, select **Config Lists**. The MEASURE CONFIGURATION LISTS screen is displayed.
3. Select a measure configuration list.

The number of indexes is displayed below the index list. To go to a specific index, you can use the Jump to Index option.

Using SCPI commands:

To view the number of configuration indexes in a measure configuration list named MyMeasList, send the following command:

```
:SENSe:CONFiguration:LIST:SIZE? "MyMeasList"
```

Using TSP commands:

To view the number of configuration indexes in a measure configuration list named MyMeasList, send the following command:

```
dmm.measure.configlist.size("MyMeasList")
```

Save a configuration list

Configuration lists are removed when you turn the instrument off and turn it on again or if you reset the instrument.

To save a configuration list, create a configuration script. A configuration script saves the settings of the instrument, including all configuration lists. You can do this using any of the following:

- Front panel option Menu > Save Setup.
- SCPI command `*SAV`
- TSP command `createconfigscript()`

See [Saving setups](#) (on page 4-92) for additional information.

Remote commands for configuration list operations

You can use the following remote commands to create and maintain configuration lists.

Action	SCPI command
	TSP command
Create a configuration list	[:SENSe[1]]:CONFfiguration:LIST:CREate (on page 12-179) dmm.measure.configlist.create() (on page 14-214)
Restore the settings in a configuration list to the instrument	[:SENSe[1]]:CONFfiguration:LIST:RECall (on page 12-182) dmm.measure.configlist.recall() (on page 14-217)
View the contents of a configuration list index as TSP commands	[:SENSe[1]]:CONFfiguration:LIST:QUERy? (on page 12-181) dmm.measure.configlist.query() (on page 14-216)
Delete a configuration list or an index in a configuration list	[:SENSe[1]]:CONFfiguration:LIST:DELeTe (on page 12-179) dmm.measure.configlist.delete() (on page 14-215)
View available configuration lists	[:SENSe[1]]:CONFfiguration:LIST:CATalog? (on page 12-178) dmm.measure.configlist.catalog() (on page 14-213)
Determine the number of configuration indexes in a configuration list	[:SENSe[1]]:CONFfiguration:LIST:SIZE? (on page 12-183) dmm.measure.configlist.size() (on page 14-218)
Save a configuration list	[:SENSe[1]]:CONFfiguration:LIST:STORe (on page 12-183) dmm.measure.configlist.store() (on page 14-218)
Store settings for a function into a configuration list regardless of the active state of the function	No SCPI version dmm.measure.configlist.storefunc() (on page 14-220)
Set up functions regardless of the active state of the function so they can be saved using <code>dmm.measure.configlist.storefunc()</code>	No SCPI version dmm.measure.setattribute() (on page 14-299)

Settings stored in a measure configuration index

The following measure settings can be stored in a configuration index.

Function settings:

- Analog Trigger Mode
- Analog Trigger Edge Level
- Analog Trigger Edge Slope
- Analog Trigger Window Direction
- Analog Trigger Window High Boundary
- Analog Trigger Window Low Boundary
- Aperture
- Auto Delay
- Auto Zero
- Autorange
- Bias Level
- Count
- dBm Reference
- Decibel Reference
- Detector Bandwidth
- Display Digits
- Function
- Input Impedance
- Line Sync
- NPLC
- Offset Compensation
- Range
- Open Lead Detector
- RTD - Four-Wire
- RTD - Three-Wire
- RTD - Two-Wire
- RTD Alpha
- RTD Beta

- RTD Delta
- RTD Zero
- Reference Junction
- Simulated Reference Junction
- Threshold Autorange
- Threshold Level
- Threshold Range
- Thermistor
- Thermocouple
- Transducer
- Unit
- User Delays 1 to 5

Filter settings:

- Enable
- Count
- Type
- Window

Limit 1 and Limit 2 settings:

- Enable
- Audible
- Auto Clear
- High Value
- Low Value

Math settings:

- Enable
- Format
- b(Offset)
- m(Scalar)
- Percent

Relative offset settings:

- Enable
- Level
- Method (DC Ratio function)

Digitize settings stored in a measure configuration list index

The following digitize settings are stored in a configuration index.

- Digitize Function
- AC Coupling Filter
- AC Coupling Frequency
- Analog Trigger Mode
- Analog Trigger Edge Level
- Analog Trigger Edge Slope
- Analog Trigger Window Direction
- Analog Trigger Window High Boundary
- Analog Trigger Window Low Boundary
- Aperture
- Count
- dBm Reference
- Decibel Reference
- Display Digits
- Input Impedance
- Range
- Sample Rate
- Signal Coupling
- Unit
- User Delays 1 to 5

Limit 1 and Limit 2 settings:

- Enable
- Audible
- Auto Clear
- High Value
- Low Value

Math settings:

- Enable
- Format
- b(Offset)
- m(Scalar)
- Percent

Relative offset settings:

- Enable
- Level

Switching and scanning

In this section:

Switching modules	5-1
Channel naming	5-6
Set up measurement channels.....	5-7
Making measurements with channels	5-14
Opening and closing channels	5-15
Channel status	5-24
Relay closure counts	5-27
Using watch channels	5-28
Scanning and triggering	5-29
Multiple-channel operation	5-57

Switching modules

This section describes how to install, connect, control, and make measurements with switching modules.

You can use the following switching module with the DAQ6510:

- 7700: 20-channel Differential Multiplexer Module
- 7701: 32-channel Differential Multiplexer Module
- 7702: 40-channel Differential Multiplexer Module
- 7703: 32-channel High Speed Differential Multiplexer Module
- 7705: 40-channel Single-pole Control Module
- 7706: 20-channel Differential Multiplexer with Automatic CJC, 16 Digital Outputs, 2 Analog Outputs, and Counter/Totalizer
- 7707: 32-channel Digital I/O Module with 10-channel Differential Multiplexer
- 7708: 40-channel Differential Multiplexer Module with Automatic CJC
- 7709: 6x8 Matrix Module
- 7710: 20-channel Solid-state Differential Multiplexer with Automatic CJC
- 7711: 2 GHz 50 Ohm RF Module
- 7712: 3.5 GHz 50 Ohm RF Module

The 7700 and 7702 modules provide two additional channels that support current measurements that use the DAQ6510 current measure functions. Refer to the switching module datasheets for the current carry specifications.

The 7705, 7711, and 7712 provide signal switching only and no direct measurement using the DAQ6510.

Refer to the switching module instruction sheets to determine which module is appropriate for your application.

Pseudocards

You can perform most open, close, and scan operations and configure your system without having an actual switching module installed in your instrument. Using the remote interface, you can assign a pseudocard to an empty card slot, allowing the instrument to operate as if a switching module were installed.

NOTE

While most operations can be simulated with a pseudocard, some operations, such as channel delays, cannot be simulated.

A pseudocard cannot be configured from the front panel. However, once the remote configuration is complete, you can use the front panel to use the pseudocard.

When the instrument is turned off, the pseudocard information is cleared. To preserve the settings, pseudocard information can be saved in a setup or a script. Refer to [Saving setups](#) (on page 4-92) for information.

There is a pseudocard for every Keithley Instruments Model 77XX series switching module.

To install pseudocards using SCPI commands, refer to the command description for [:SYSTem:PCARd<slot>](#) (on page 12-226).

To install pseudocards using TSP commands, refer to the command description for [slot\[slot\].pseudocard](#) (on page 14-419).

Connections to a switching module

WARNING

Connection information for switching modules is intended for qualified service personnel only, as described by the types of product users in the [Safety precautions](#). Do not attempt to connect a device under test (DUT) or external circuitry to a switching module unless qualified to do so.

WARNING

To prevent electric shock that could result in serious injury or death, observe the following safety precautions. Before making or breaking connections to the switching module, make sure the DAQ6510 is turned off and power is removed from all external circuitry. Do not connect signals that will exceed the maximum specifications of the switching module. For maximum specifications, refer to the specifications for the switching module, which are available at tek.com/keithley (tek.com/products/keithley).

WARNING

If the front-panel terminals and the switching module terminals are connected at the same time, the test-lead insulation must be rated to the highest voltage that is connected. For example, if 1000 V is connected to the front-panel input, the test-lead insulation for the switching module must also be rated for 1000 V. Failure to use test leads rated for at least 1000 V can result in injury, death, or instrument damage.

Refer to the documentation for your switching module for connection and switching module installation information.

Identify the installed switching module

You can identify the switching module that is installed in the DAQ6510 from the front panel or by using remote commands.

From the front panel:

1. Select **MENU**.
2. Under Channel, select **Control**.

The switching modules that are installed are displayed at the top of the screen.

The front-panel display does not indicate if the switching module is a pseudocard.

Using SCPI commands, send:

```
:SYSTem:CARD1:IDN?  
:SYSTem:CARD2:IDN?
```

This returns information about the modules in the slots, in a format similar to:

```
7706,25Ch ALL-IN-ONE w/DIO,01.40e,1243657  
7700,Pseudo 20Ch Mux w/CJC,???????,???????????
```

Pseudocards have `Pseudo` as part of the return.

Using TSP commands, send:

```
print(slot[1].idn)  
print(slot[2].idn)
```

This returns information about the modules in the slots, in a format similar to:

```
7706,25Ch ALL-IN-ONE w/DIO,01.40e,1243657  
7700,Pseudo 20Ch Mux w/CJC,???????,???????????
```

Pseudocards have `Pseudo` as part of the return.

Determine module capabilities

If you are communicating with your DAQ6510 remotely, you can use SCPI or TSP commands to determine which channels of the installed switching module, if any, support specific features.

The features and the commands to query them are listed in the following table.

SCPI	TSP
Current measurements	
:SYSTem:CARD<slot>:ACHannel[:START]? (on page 12-198) :SYSTem:CARD<slot>:ACHannel:END? (on page 12-199)	slot[slot].amps.startchannel (on page 14-410) slot[slot].amps.endchannel (on page 14-409)
Analog outputs	
:SYSTem:CARD<slot>:AOUTput[:START]? (on page 12-200) :SYSTem:CARD<slot>:AOUTput:END? (on page 12-201)	slot[slot].analogoutput.startchannel (on page 14-411) slot[slot].analogoutput.endchannel (on page 14-411)
Commonside 4-wire ohm channels	
:SYSTem:CARD<slot>:CSOhms? (on page 12-202)	slot[slot].commonsideohms (on page 14-412)
Digital inputs and outputs	
:SYSTem:CARD<slot>:DIGitalio[:START]? (on page 12-203) :SYSTem:CARD<slot>:DIGitalio:END? (on page 12-204)	slot[slot].digitalio.startchannel (on page 14-413) slot[slot].digitalio.endchannel (on page 14-413)
Isolated channels	
:SYSTem:CARD<slot>:ISOLated[:START]? (on page 12-206) :SYSTem:CARD<slot>:ISOLated:END? (on page 12-207)	slot[slot].isolated.startchannel (on page 14-416) slot[slot].isolated.endchannel (on page 14-416)
Matrix channels	
:SYSTem:CARD<slot>:MATRix:COLumns? (on page 12-208) :SYSTem:CARD<slot>:MATRix[:ROWS]? (on page 12-208)	slot[slot].matrix.columns (on page 14-417) slot[slot].matrix.rows (on page 14-418)
Totalizer channels	
:SYSTem:CARD<slot>:TCHannel[:START]? (on page 12-209) :SYSTem:CARD<slot>:TCHannel:END? (on page 12-210)	slot[slot].totalizer.startchannel (on page 14-422) slot[slot].totalizer.endchannel (on page 14-421)

Temperature sensor channels	
:SYSTem:CARD<slot>:TEMPsensor? (on page 12-211)	slot[slot].tempsensor (on page 14-421)
Voltage or two-wire measurements	
:SYSTem:CARD<slot>:VCHannel[:START]? (on page 12-212) :SYSTem:CARD<slot>:VCHannel:END? (on page 12-213)	slot[slot].voltage.startchannel (on page 14-423) slot[slot].voltage.endchannel (on page 14-423)
Maximum voltage of all channels	
:SYSTem:CARD<slot>:VMAX? (on page 12-214)	slot[slot].maxvoltage (on page 14-419)

Channel naming

In the DAQ6510, channels are named using the slot number and the channel number. The slot number is the number of the slot on the rear panel where the card is installed. The channel assignment is formatted as `SCH`, where:

- `S` is the slot number
- `CH` is a two-digit channel number

For example:

- `101` = slot 1, channel 1
- `210` = slot 2, channel 10

Set up measurement channels

Each switching module contains connections to devices under test or external equipment. These connections are called channels.

The features of each channel determine the types of actions you can apply to that channel. For information on the types of channels, refer to the documentation for the switching module.

You can set up channels using the front-panel interface or remote commands.

When you are using the front-panel interface, you set up channels and scans through the Channel Settings, Channel Control, and Channel Scan menus. You can also work with channels from the home screen through the CHANNEL and SCAN swipe screens.

You can set up measurement channels for DMM functions and apply calculations and filters to the measurements. You can also set up options specific to channels, such as assigning labels and adding channel delays.

The following topics describe how to set up channels for measurements from the front panel and using remote commands.

Set up measurement channels using the front panel

To set up measurement channels, use the Channel Settings screen. If two cards or pseudocards are installed, use the Slot button at the top of the screen to select the card that contains the channels. You can select channels from both cards. The channels that are being edited are listed to the right of the Slot button.

You select channels using the channel selections in the left pane of the screen. To select all switching channels, select **All**. If you select a slot that contains a card that supports mixed channel types, such as the 7706 or 7707, the All option only selects and clears switch channels. If you select a nonswitch channel, such as a digital I/O, DAC, or totalizer channel, All is not available.

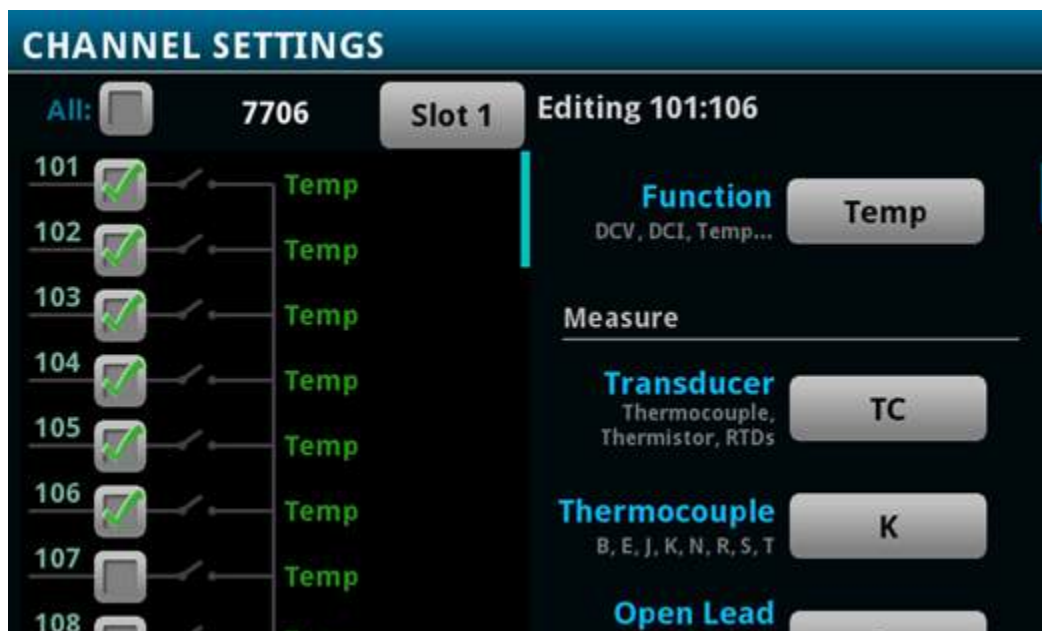
In the right pane, you can set the function, channel label, and channel delay. After setting the function, you can set measure settings and calculations for the function.

You can work with multiple channels if the channels have the same function applied. If a setting is different for the selected channels, "Various" is displayed for the setting.

To set up channels through the front panel:

1. Make sure the front-panel Terminals switch is set to **Rear**.
2. Press the **MENU** key.
3. Under Channel, select **Settings**.
4. Verify that the correct slot is selected.
5. Select a channel or multiple channels. Settings you make apply to all the channels that are selected.
6. Use the options on the right side of the screen to make the settings. You can set:
 - **Measure options:** Refer to [Measure Settings menu](#) (on page 3-38) for descriptions of options.
 - **Calculations:** Refer to [Measure Calculations menu](#) (on page 3-48) for descriptions of the math and filter options.
 - **Labels:** Refer to [Assign labels to channels](#) (on page 5-13) for detail.
 - **Channel Delay:** Refer to [Add a channel delay](#) (on page 5-14) for detail.

Figure 96: Channel Settings screen



Set up channels using SCPI commands

If you are sending commands from a remote interface, you include the three-digit channel assignment in the channel list parameter for the commands.

In SCPI commands, the first channel number in a command is prefaced by @. To designate multiple individual channels, separate the channels with commas. In the following example, the command will close channels 1 and 3 on the card in slot 1.

```
ROUT:CLOS (@101, 103)
```

To designate a range of channels, separate the first and last channel number with a colon. The range can be from lowest to highest numbered channel or highest to lowest. The range can include channels from both slots. The following example sets DMM settings on a range of twenty channels and then creates a scan with those channels using the DC voltage function.

```
SENS:FUNC "VOLT", (@101:120)  
SENS:VOLT:NPLC 0.1, (@101:120)  
ROUT:SCAN:CRE (@101:120)  
INIT
```

Some commands allow you to set all channels in a slot or all channels in the instrument. In this case, you can send `SLOT1` to set all channels in slot 1; `SLOT2` for all channels in slot 2, or `ALLSLOTS` for all channels. For example, to set all channels on slot 1 to measure voltage, you can send:

```
SENS:FUNC "VOLT", (@SLOT1)
```

When you send a range that includes both slots, `SLOT1`, `SLOT2`, or `ALLSLOTS` to a slot that contains channels that cannot accept that setting, the instrument generates an error, but the change is made to all channels for which the setting is allowed.

For example, to set up the channels in a DAQ6510 with a 7700 switching module installed to:

- Set the commands to their default states.
- Have channel 101 measure AC voltage with a low-end bandwidth of 30 Hz.
- Have channels 102 to 110 measure DC voltage using the default values.
- Have channel 121 to measure AC current.
- Have channel 122 to measure DC current.

You send:

```
*RST
SENS:FUNC 'VOLT:AC', (@101)
SENS:VOLT:AC:DET:BAND 30, (@101)
SENS:FUNC 'VOLT:DC', (@102:110)
SENS:FUNC 'CURR:AC', (@121)
SENS:FUNC 'CURR:DC', (@122)
```

For a full list of command descriptions, refer to the [SCPI command reference](#) (on page 12-1).

Set up channels using TSP commands

If you are sending commands from a remote interface, you include the three-digit channel name, `slot1`, `slot2`, or `allslots` in the channel list parameter for the commands.

In TSP commands, the channel numbers are in quotes. To designate multiple individual channels, separate the channels with commas. In the following example, the command opens channels 1 and 3 on the card in slot 1.

```
channel.open("101, 103")
```

To designate a range of channels, separate the first and last channel number with a colon. The range can be from lowest to highest numbered channel or highest to lowest. The following example sets channel 1 to 9 on slot 1 to the DC voltage measurement function and then creates a scan with those channels.

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)  
scan.create("101:109")
```

Some commands allow you to set all channels in a slot or all channels in the instrument. In this case, you can send `slot1` to set all channels in slot 1, `slot2` for all channels in slot 2, or `allslots` for all channels in all slots. For example, to set all channels on slot 1 to measure voltage, you can send:

```
channel.setdmm("allslots", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
```

When you send `slot1`, `slot2`, or `allslots` to a slot that contains channels that cannot accept that setting, the instrument generates an error, but the change is made to all channels for which the setting is allowed.

For example, to set up the channels in a DAQ6510 with a 7700 switching module installed to:

- Set the commands to their default states.
- Have channel 101 measure AC voltage with a low-end bandwidth of 30 Hz.
- Have channels 102 to 110 measure DC voltage using the default values.
- Have channel 121 to measure AC current.
- Have channel 122 to measure DC current.

You send:

```
reset()  
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_AC_VOLTAGE)  
channel.setdmm("101", dmm.ATTR_MEAS_DETECTBW, dmm.DETECTBW_30HZ)  
channel.setdmm("102:110", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)  
channel.setdmm("121", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_AC_CURRENT)
```

```
channel.setdmm("122", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_CURRENT)
```

For a full list of command descriptions, refer to the [TSP command reference](#) (on page 14-1).

Assign labels to channels

You can assign labels to channels. The label name is displayed on the home screen and the Channel swipe screen when the channel is displayed. On the Graph and Histogram screens, you can use the label when selecting traces. If you are using remote commands, you can use the label instead of a channel number in commands.

The label must be unique; you cannot assign the same label to more than one channel. Labels cannot start with a digit. They can be up to 19 characters. On the front panel of the instrument, only the first few characters are displayed.

Labels are reset when the instrument is reset and when power is cycled.

To set up labels from the front panel:

1. Under Channel, select either **Settings** or **Scan**.
2. Select a single channel.
3. Scroll to the bottom of the measure options.
4. Set the **Label**.

To set up channels using SCPI commands, send the command:

```
:ROUTe[:CHANnel]:LABel "<label>", (@<channel>)
```

Where `<label>` is the channel identifier and `<channel>` is the channel number. For example, to set channel 101 to `test2`, send:

```
:ROUTe:LABel "test2", (@101)
```

To clear a label, set it to an empty string:

```
:ROUTe:LABel "", (@101)
```

To set up channels using TSP commands, send the command:

```
channel.setlabel("channelNumber", "labelname")
```

Where `labelname` is the channel identifier and `channelNumber` is the channel number. For example, to set channel 101 to `test2`, send:

```
channel.setlabel("101", "test2")
```

To clear a label, set it to an empty string:

```
channel.setlabel("101", "")
```

Add a channel delay

You can set a channel delay to occur after the relay closes. This allows extra settling time for the relay. This delay is in addition to normal settling time.

From the front panel:

1. Under Channel, select either **Settings** or **Scan**.
2. Select the channel or channels.
3. Scroll to the bottom of the measure options.
4. Set the **Channel Delay**.

Using SCPI commands, send the command:

```
:ROUTe[:CHANnel]:DElay <delay>, (@<channelList>)
```

Where <delay> is the delay time and <channelList> is the channel or list of channels. For example, to set all channels on slot 1 to have a 0.1 s delay, send:

```
:ROUTe:DElay 0.1, (@slot1)
```

Using TSP commands, send the command:

```
channel.setdelay(channelList, delay)
```

Where *channelList* is the channel or list of channels and *delay* is the delay time. For example, to set all channels on slot 1 to have a 0.1 s delay, send:

```
channel.setdelay("101", 0.1)
```

Making measurements with channels

When a channel has a measurement function assigned to it, you can use the DMM functions of the DAQ6510 to make measurements on that channel.

You can make individual measurements or include the channel in a scan. The following topics describe how to make an individual measurement. To use a scan, refer to [Scanning and triggering](#) (on page 5-29).

CAUTION

To prevent damage to a switching module, do not exceed the maximum signal level input for that switching module. Most switching modules are rated for 303 V. To query the maximum voltage, see [:SYSTem:CARD<slot>:VMAX?](#) (on page 12-214) or [slot\[slot\].maxvoltage](#) (on page 14-419).

Make measurements from the front panel

To make a measurement from the front panel, close the channel. If the measurement trigger type is set to continuous, the DAQ6510 starts making measurements on that channel and stores them in the selected reading buffer. If the measurement trigger type is set to Manual Trigger Mode, measurements are made when you press the front-panel TRIGGER key.

If you close a measurement channel and the channel is not set up for a function, you are prompted to select a function.

Make channel measurements using SCPI commands

Use the `READ?` command to make a reading from a closed channel.

For example, to set up channel 101 to the DC voltage function, close the channel, and make a reading, send:

```
*RST
SENSe:FUNCTion "VOLTage", (@101)
ROUTe:CLOSe (@101)
READ?
```

Make channel measurements using TSP commands

Use the `dmm.measure.read()` command to make a reading from a closed channel.

For example, to set up channel 101 to the DC voltage function, close the channel, and make a reading, send:

```
reset()
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.close("101")
print(dmm.measure.read())
```

Opening and closing channels

You can open and close channels from the front panel CHANNEL swipe screen or the CHANNEL CONTROL screen. You can also use remote commands to open and close channels.

The actions when a channel is opened or closed depend on the switching module operation and the type of channel.

Operation of switching modules

A switching module lets you switch among a number of input signals to the DAQ6510. The channel control and scanning capabilities depend on the capabilities of the switching module. Refer to the documentation supplied with the switching module for specific connection information.

The action of the close command depends on which, if any, function is set for the channel.

If you are working with a measure channel, when a measure function is assigned to a channel and that channel is closed, the backplane and any channels that are paired to that channel are also closed. All other channels or channel pairs that could affect the measurements are opened. Before a measurement is made, settling time and any user-specified delays are added.

When a channel is opened and a measure function is assigned, the backplane and any channels paired to that channel are also opened. Only one channel or channel pair can be closed at a time. This channel is displayed on the CHANNEL swipe screen and the Channel Control screen. The paired channel is shown on the Channel Control screen. Backplane channels make the connections to the DAQ6510 DMM. For a 2-wire function, when a measure channel is closed, the backplane channel closes to connect the system channel to the measure input HI and LO. For a 4-wire function, the sense backplane isolation channel is also closed to make the sense connections to the DAQ6510.

If you did not assign a function to a channel and that channel is opened or closed, channel open or close does not affect the open or close state of any other channel. You can select multiple channels.

Connection methods for close operations

You can dictate the order in which relays are opened and closed using the channel connection rule.

WARNING

With no connection rule, the instrument attempts to simultaneously open and close switch channels in order to minimize the command execution time. This is the default behavior. 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. 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.

When the connection rule is set to break before make, 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. 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 make before break, 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.

Cold switching is highly recommended.

CAUTION

Hot switching can dry weld reed relays such that they will always be on. Hot switching is recommended only when external protection is provided.

The channel connect rule determines the order in which multiple channels are opened and closed on the instrument. This attribute applies to electromechanical, reed, and solid-state relay switching cards.

You can set the channel connect rule to be:

- **BBM** (break before make): The instrument ensures that all switch channels open before any switch channels close. It is used to avoid momentary shorting of two voltage sources.
- **MBB** (make before break): The instrument ensures that all switch channels close before any switch channels open. It is used to eliminate transients caused by switching between current sources. MBB should be applied with caution because it connects two test devices together for the duration of the switch close settle time.
- **CONCURRENT**: Permits the instrument to initiate close and open operations simultaneously. This minimizes settling time for the close operation. This is the default.

NOTE

You cannot guarantee the sequence of open and closure operations when the channel connect rule is set to CONCURRENT (the default). It is highly recommended that you implement cold switching when the channel connect rule is set to CONCURRENT.

To set the channel connect rule through SCPI commands:

Use the [ROUTE:\[CHANnel\]:CONNection:RULE](#) (on page 12-65) command.

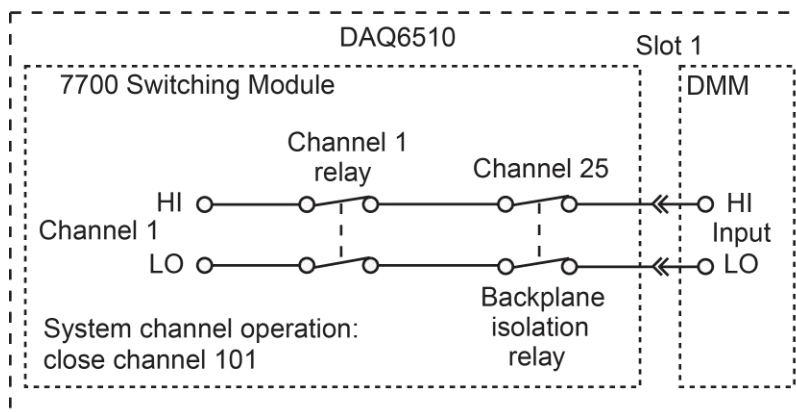
To set the channel connect rule through TSP commands:

Use the [channel.connectrule](#) (on page 14-69) command.

Two-wire functions

The following figure shows an example of how a channel is connected to the DMM input of the DAQ6510. This example shows a 7700 switching module installed in slot 1 of the DAQ6510. When channel 101 is closed, both the channel 1 relay and the backplane isolation relay (channel 25) close to connect the channel to the DMM.

Figure 97: 2-wire channel connections to the DAQ6510

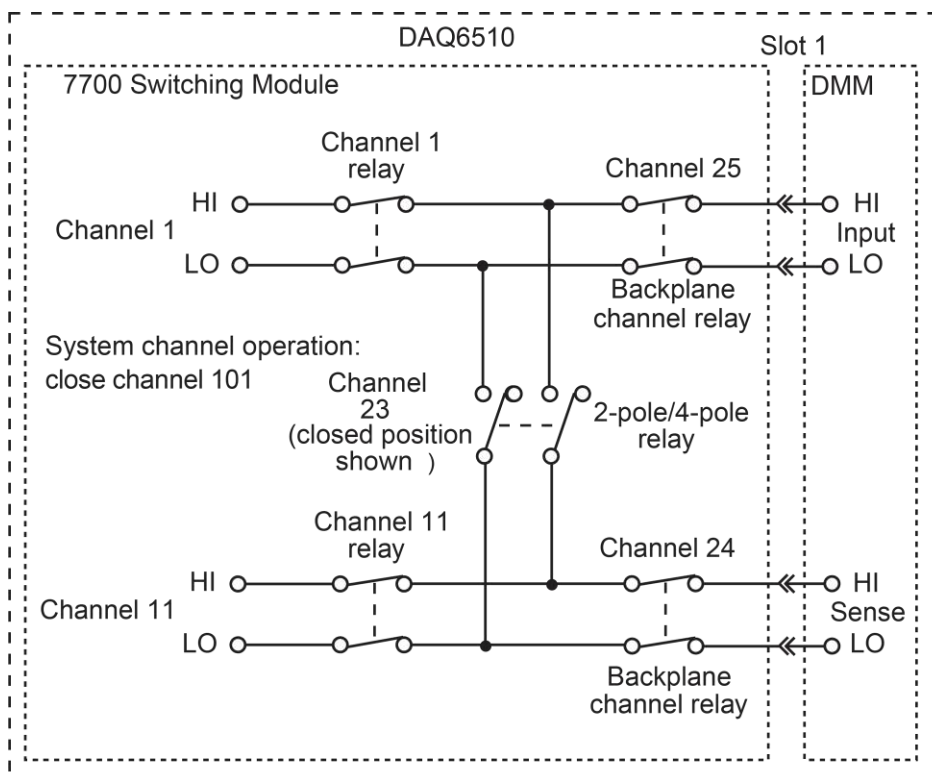


Four-wire functions (paired channels)

A 4-wire function, such as 4-wire ohms, requires that measurement channels be paired. For example, if the switching module has 20 measurement channels, channels 1 through 10 are paired to channels 11 through 20. Channel 1 is paired to channel 11, channel 2 is paired to channel 12, channel 3 is paired to channel 13, and so on.

The following figure shows an example of channel connections for a 4-wire function. In this example, a 7700 switching module is installed in slot 1 of the mainframe, and a 4-wire function is selected. When channel 101 is closed, the channel 1 relay and the input backplane isolation relay (channel 25) closes to connect the channel to DMM input. The channel 11 relay and the sense backplane channel relay (channel 24) close to connect the paired channel to DMM sense. Also note that the channel 23 relay closes to isolate channel 1 from channel 11.

Figure 98: 4-wire channel connections to DAQ6510



Refer to your switching module documentation for information on how channels are paired for other switching modules.

Controlling channels using the front-panel CHANNEL CONTROL screen

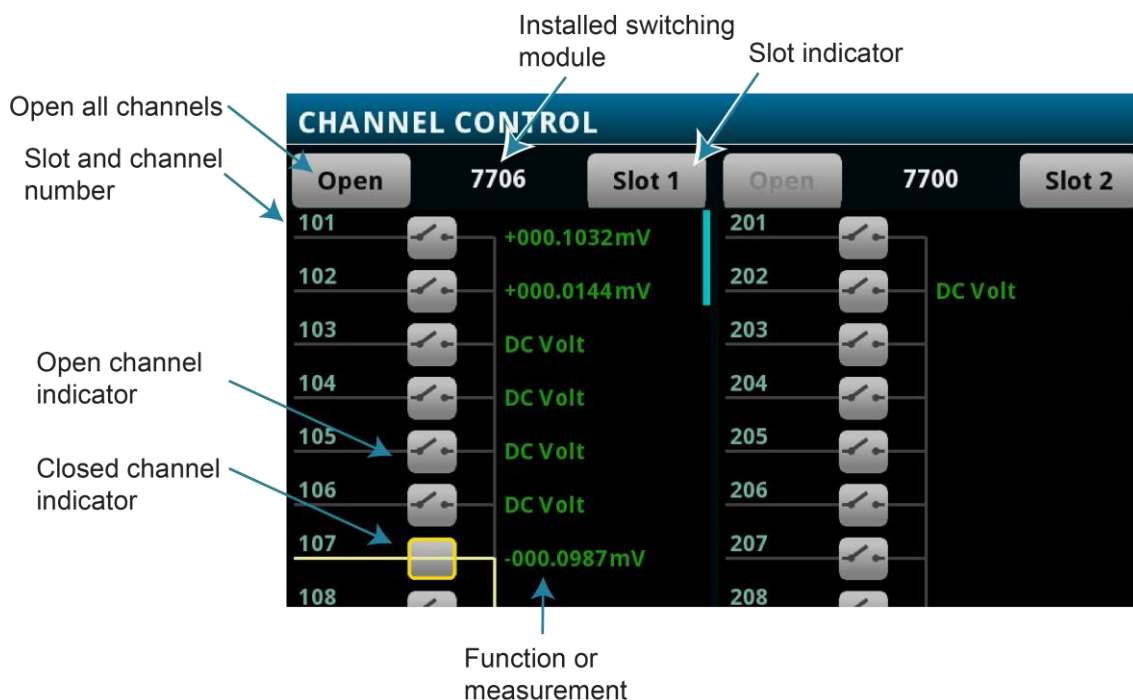
The action when you close a channel using the front panel depends on the channel type. If the channel is a measurement channel and the trigger measurement method is set to Continuous Measurement, the instrument starts making measurements when the channel is closed. If the trigger measurement method is set to Manual Trigger Mode, the instrument makes measurements when the channel is closed and the front-panel TRIGGER key is pressed.

Before a measurement channel is closed, the previous closed channel or channel pair is opened. If you are closing a measurement channel and no function is defined for the channel, you are prompted to select the function.

If you are working with a switch-only channel, the channel is closed. Other channels are not opened automatically.

The CHANNEL CONTROL screen includes options to control channels, as shown in the following figures. Descriptions of the options follow the figures.

Figure 99: CHANNEL CONTROL screen with measure channels



Open button: Opens all channels. Only available if there are closed channels.

Card number: The model number of the card or pseudocard.

Slot indicator: Selects the slot that is displayed on that pane of the screen. You can display the same slot in both panes. This allows you show different sets of channels on the same switching module.

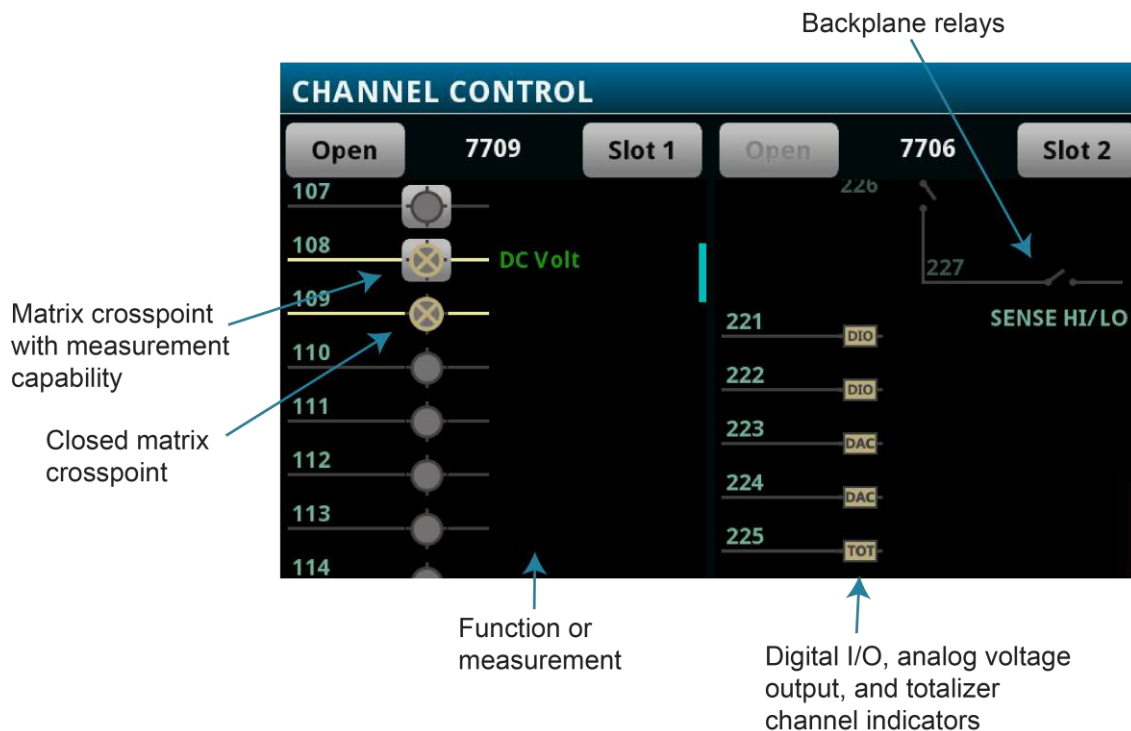
Slot and channel number: The channel number, which includes the slot number. If a label is assigned, the label is displayed below the line.

Open channel indicator: Select this icon to close a channel. If channel is paired, when you close one relay, both are shown as closed. If you attempt to close a relay for a measure channel and no function is assigned to that relay, you are prompted to assign a measure function.

Closed channel indicator: Select this icon to open a channel.

Function or measurement: When a measure function is assigned to a channel, but no measurement was made, the name of the measure function is displayed. If a measurement was made, the most recent measurement value is displayed.

The CHANNEL CONTROL screen also includes options to control matrix channels, view backplane relay status, and show nonmeasurement channels, as shown in the following figure. Descriptions of the options follow the figure.

Figure 100: CHANNEL CONTROL with backplanes and matrix channels

Backplane relays: Displays the status of SENSE HI/LO, INPUT HI/LO, and backplane relays. You cannot directly control these relays from the front panel when the Channel Access on the instrument is set to Simple. You can control these relays from the front panel when Channel Access is set to Full. See [Multiple-channel operation](#) (on page 5-57) for information on using Channel Access.

Matrix crosspoint with measurement capability: A matrix crosspoint that can be used to make measurements.

Closed matrix crosspoint: A closed matrix crosspoint that is switch-only.

DIO: The channel is a digital I/O channel.

DAC: The channel is an analog voltage output channel.

TOT: The channel is a totalizer channel. Totalizer channels are read-only.

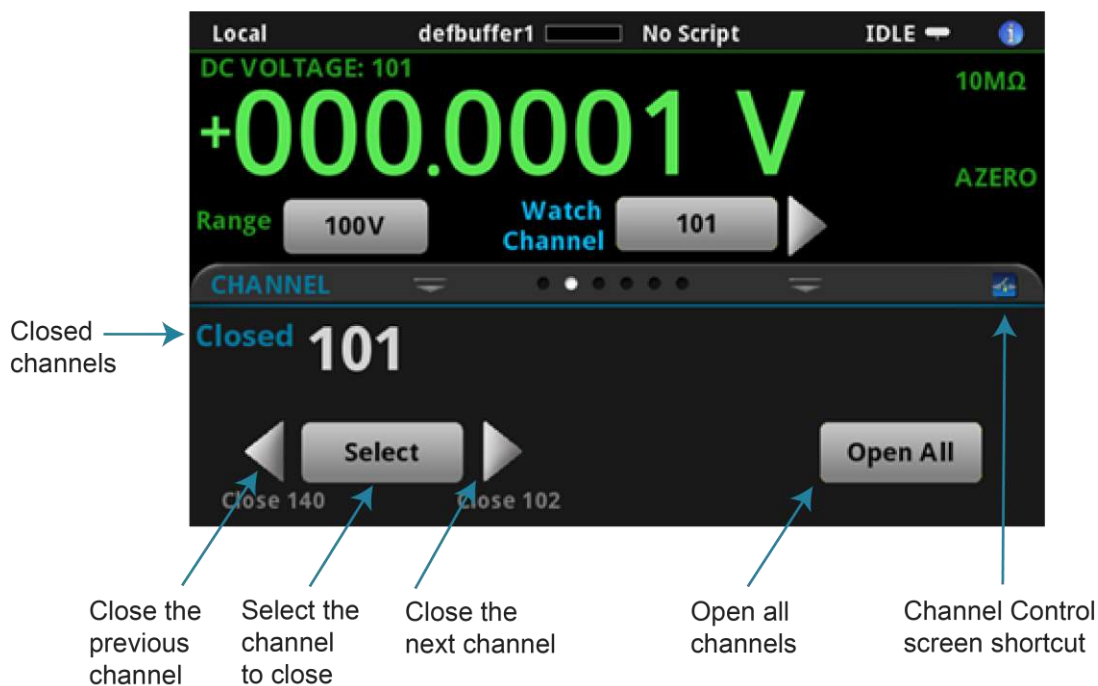
NOTE

The on-screen text for each DIO channel indicates the correct minimum and maximum width for the DIO channel if it is the only DIO channel that is selected. However, if multiple DIO channels are selected, these values display the lowest DIO widths.

Controlling channels using the front-panel CHANNEL swipe screen

The CHANNEL swipe screen allows you to select, open, and close channels, as shown in the following figure. Descriptions of the options follow the figure.

Figure 101: CHANNEL swipe screen options



The CHANNEL swipe screen has the following options:

- **Channel Control shortcut:** Opens the CHANNEL CONTROL menu, which you can use to choose functions for measurement channels and open and close channels.
- **Closed channels:** Displays the channel numbers of closed channels. If the list of channels does not fit on the screen, select the ... button to display a dialog box with a full list of closed channels.
- **Close the previous channel:** Closes the previous channel.
- **Close the next channel:** Closes the next channel.
- **Open All:** Opens all channels. This button is always available.
- **Select:** Select the channel to close.

Control channels from a remote interface

The commands to close and open system channels are listed in the following table.

The SCPI commands for closing and opening channels include:

- [:ROUTe\[:CHANnel\]:CLOSE](#) (on page 12-61): This command closes the channels and channel pairs that are specified by the channel list parameter.
- [:ROUTe\[:CHANnel\]:OPEN](#) (on page 12-76): This command opens the specified channels and channel pairs.
- [:ROUTe\[:CHANnel\]:OPEN:ALL](#) (on page 12-77): This command opens all channels on all slots, including nonmeasurement channels.

The TSP commands for closing and opening channels include:

- [channel.close\(\)](#) (on page 14-66): This command closes the channels and channel pairs that are specified by the channel list parameter.
- [channel.getclose\(\)](#) (on page 14-70): This command queries for closed channels.
- [channel.open\(\)](#) (on page 14-83): This command opens the specified channels and channel pairs.

Channel status

To determine if channels are closed or open, you can view their status on the front panel or use remote commands.

To check channel status on the front panel:

1. Select the **Home** key.
2. Go to the **CHANNEL** swipe screen. Closed channels are displayed on the screen.

For a more complete view of channels, select the **MENU** key and select Channel **Control**. Closed relays are displayed on the screen, as shown in [Controlling channels from the front panel](#) (on page 5-20).

Using SCPI commands, send the command:

```
:ROUTe[:CHANnel]:STATe? (@<channelList>)
```

Where <channelList> is the list of channels.

For example, to get the status of channels 101 to 109, send:

```
:ROUTe:STATe? (@101:109)
```

The status of all channels on all slots is returned. The return is similar to:

```
1,0,0,0,0,0,0,1,0
```

This return indicates that the first and eighth channels (101 and 108) are closed and all other channels are open.

To return only the closed channels, send:

```
:ROUTe:CLOSe? (@101:109)
```

The return is similar to:

```
(@101,108)
```

Indicating that channels 101 and 108 are closed.

Using TSP commands, send the command:

```
print(channel.getstate("channelList"))
```

Where *channelList* is the list of channels.

For example, to get the status of channels 101 to 109, send:

```
print(channel.getstate("101:109"))
```

The return is similar to:

```
[1]=channel.IND_CLOSED, [2]=0, [3]=0, [4]=0, [5]=channel.IND_CLOSED, [6]=0, [7]=0,  
[8]=0, [9]=0
```

This return indicates that the first and fifth channels in the array (channels 101 and 105) are closed and all other channels are open.

To return only closed channels, send:

```
print(channel.getclose("channelList"))
```

For example, to get the status of channels 101 to 109, send:

```
print(channel.getclose("101:109"))
```

The return is similar to:

```
[1]=101, [2]=105
```

This return indicates that channels 101 and 105 are closed.

Relay closure counts

The DAQ6510 keeps an internal count of the number of times each relay has been closed. This count can help you determine when relays require replacement. Refer to the switching module documentation for the contact life specifications for the relays.

You can also set how often closure counts are stored in nonvolatile memory. If the DAQ6510 is turned off before the count is stored, the counts are lost. Storing them more frequently can prevent loss of the counts, but can also result in loss of scan throughput.

Relay closure counts are updated in temporary memory when a channel is closed regardless of how it was closed. The DAQ6510 stores relay closure counts in nonvolatile memory when counts are queried and at a preset interval.

To get relay closure counts, use the following commands (counts are not available from the front panel):

- SCPI: [:ROUTe\[:CHANnel\]:CLOSE:COUNT?](#) (on page 12-64)
- TSP: [channel.getcount\(\)](#) (on page 14-72)

To retrieve and set the interval for nonvolatile storage (this setting is not available from the front panel):

- SCPI: [:ROUTe\[:CHANnel\]:CLOSE:COUNT:INTERval](#) (on page 12-63)
- TSP: [channel.getcountinterval\(\)](#) (on page 14-73)
- TSP: [channel.setcountinterval\(\)](#) (on page 14-86)

Using watch channels

Watch channels are channels that you want to focus attention on. Watch channels affect what you see on the CHANNEL and STATISTICS swipe screens. They also determine which readings you see on the home screen.

In the Reading Table, you can select the watch channels to filter the buffer so that only information from the watched channels is shown.

In the Graph screens, you can select the watch channels. Each channel in the watch channels list is displayed as a trace on the graph.

On the home screen, the Watch Channels option is available when rear-panel terminals are selected. You can select individual channels or all channels on a slot. You can have a maximum of 20 individually selected channels. When a scan is run, the data for the last channel in the Watch Channel list is displayed.

On the SCAN swipe screen, only data for the last channel in the Watch Channel list is displayed in the Scan Progress bar.

On the STATISTICS swipe screen, only statistics for the watch channels are displayed. If Watch Channel is set to a range of channels (not All), you can swipe to display the statistics for each watch channel.

On the Reading Table, you can set the reading buffer to Watch Channels, which filters the reading buffer data to display only data from the watched channels. Options in the menu allow you to adjust the watch channels.

On the Graph screen, you can select Watch Channels as the traces to graph. Options in the menu allow you to adjust the watch channels. Each channel in the range of Watch Channels is shown as a separate trace.

Watch channels are saved as part of a saved setup.

Scanning and triggering

Scans automate actions that you want to do consistently and repeatedly on a set of channels. Scans provide faster throughput and simpler setup than equivalent code or manual operation.

A scan is a series of steps arranged in a specific order that opens and closes switches sequentially for selected channels. Each step consists of a channel and its related channels, such as backplane channels or paired channels for 4-wire operations. When a channel is opened or closed, the related channels are also opened or closed as needed for the requested operation.

Each channel has a function and settings assigned to it. The function and settings can be different for different channels. During each step, actions occur on a channel, such as waiting for a trigger, making a measurement, or completing a step count. Scans can include channels from multiple slots.

Triggers are events that prompt the instrument to start the scan, move from one step to another in a scan, and make measurements. Triggers can come from different sources, such as a key press, digital input, or expiration of a timer. You can use triggers in the scan to synchronize actions across channels.

Channels can be set up individually or in groups. Each channel can be assigned a discrete function, function settings, calculations, and filtering.

The data that is collected from the scanned channels is stored in a reading buffer. You can also set up the scan so that data is exported to a `.csv` file at regular intervals, such as the end of a scan step or after all scans have run.

You can configure and run scans from the front panel, over a remote communications interface, or through the web interface.

NOTE

Be sure to save any existing trigger models or scans before creating a new trigger model or scan. Scans are implemented by using the trigger model. When you create a scan, a new trigger model is created. Any existing trigger models are removed. Conversely, creating a new trigger model removes the scan. For information on saving scans and trigger models, refer to [Saving setups](#) (on page 4-92).

If you change the trigger model through the Trigger Configure menu, the scan is removed from the system and must be rebuilt. Scan settings, such as the scan count and scan interval, are retained and assigned to the rebuilt scan.

Scan operation

Generally, a scan operates as follows:

1. The instrument clears the buffer that is used for storing scan data.
2. If Scan Start is set, the scan starts when the measurement or event occurs.
3. The channel action occurs after the channel delay and after the channel Start trigger is received. If the channel is making a measurement, the channel is closed. For 4-wire measurements, the paired channel is also closed. See your switching module documentation for information on which channels are paired.
4. All other channels open.
5. If a measurement is being made on a channel, the measurement is made after the measurement delay and the measurement Start trigger is received.
6. Measurements are stored in the buffer. If you do not specify a buffer, they are stored in `defbuffer1`.
7. The channel opens.
8. Channel actions repeat as set.
9. The scan repeats until the scan count is complete. If a Scan to Scan Interval is set, the next scan starts after the interval time.
10. The instrument returns to the idle state with the channels closed.

NOTE

For detailed examples of using scanning and triggering, see the *Model DAQ6510 User's Manual*, part number DAQ6510-900-01.

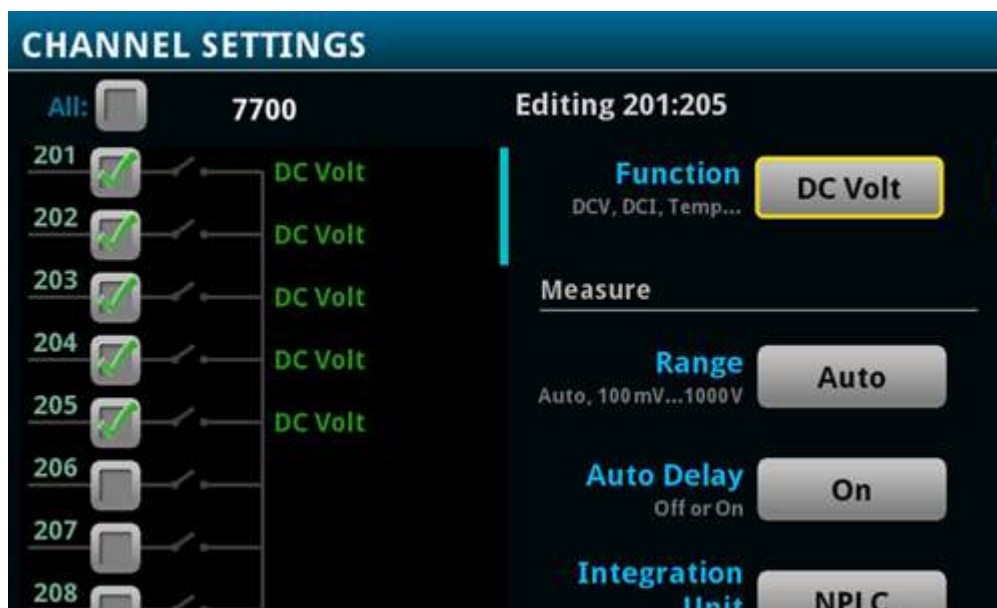
Set up the channels for the scan

You can set up channels through the front panel or remotely using SCPI commands or TSP commands.

It is best practice to configure all channel and function attributes before creating a scan. If you are unable to configure channel and function attributes before creating a scan, you can change your scan settings after starting a scan.

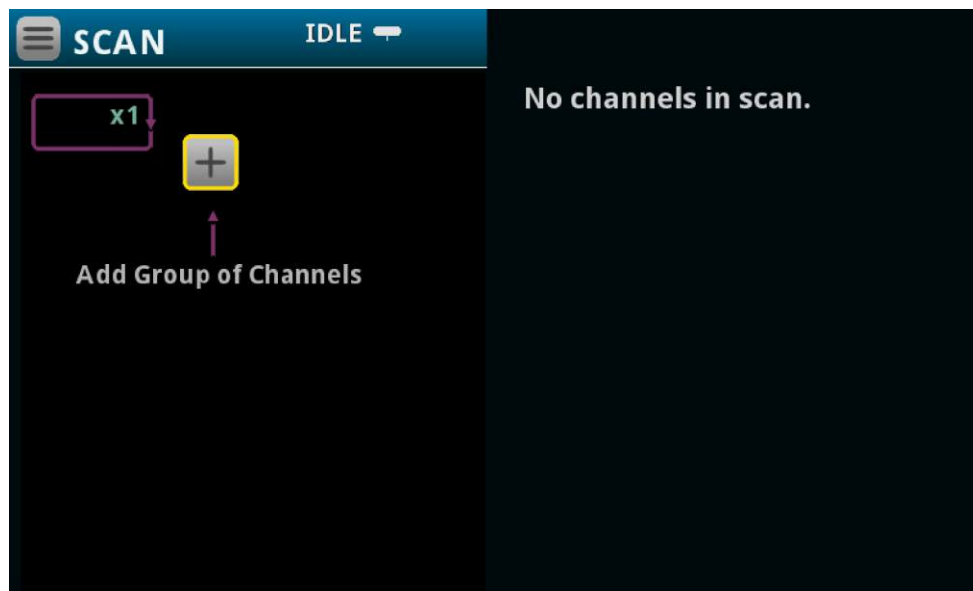
You can set up channels through the Channel Settings screen or through the SCAN screen Settings tab. You can access the CHANNEL SETTINGS screen by selecting the MENU key, then selecting the Settings icon in the Channel menu. The figure below shows the CHANNEL SETTINGS screen.

Figure 102: CHANNEL SETTINGS screen



Use the SCAN screen to configure a scan. It includes options for the channels and the scan. You can also set up triggering to establish the flow of the scan using both the DAQ6510 and external equipment. To access the SCAN screen, press **MENU**. Under Channel, select **Scan**. The figure below shows the SCAN screen.

Figure 103: SCAN settings screen



When you are setting up functions for the channels, the settings are the same as the front-panel DMM functions, including calculations. Refer to [Measure menu](#) (on page 3-37) for brief descriptions of the options that are available for each function.

If you already set up a scan and then change a function setting, the setting of the function in the scan changes, too. If the change prevents the scan from functioning properly (such as deleting a trigger referenced by the scan), the scan list may be cleared.

Some changes may cause channels to be removed from the list if they are paired with another channel for a 4-wire operation. These channels are not returned to the list if you remove 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 manually.

Create a scan

You can create scans from the front panel or through a remote interface. You can add channels to the scan in any order and use the same channel multiple times in a scan.

NOTE

After you create a scan, the Measure Configuration List screen includes the configuration list `ScanAutoCreatedMeasureList`. This list contains the settings for the scan channels. Do not modify this configuration list.

Create a scan using the front panel

To create a scan from the front panel:

1. Press the **MENU** key.
2. Under Channel, select **Scan**. If you set up channels before opening the SCAN screen, a scan preview is displayed. The preview displays the configured channels grouped by function and in channel order. If:
 - The scan preview meets your needs, select **Use Preview** to select this as your scan. The scan is created. Refer to [Set scan options](#) (on page 5-35) to continue scan setup.
 - You need a different scan order, select **Create New**. Follow the steps below to complete the scan.
3. Select **+** to add channels.
4. Select the channels for your scan from the list. You can select each channel individually, as a group, or in the order in which they should operate in the scan. If you select multiple channels, they are added to the scan in sequential order. If you need to re-order the channels, you must add them individually.
5. Adjust the function settings for the channels in the right pane.
6. Refer to [Set scan options](#) (on page 5-35) for information on the options in the Scan tab.

Create a scan using SCPI commands

In SCPI, you send a command to create the scan. You can include channels in any order in the command.

To include a range of channels, use a colon to separate the first and last channel in the list. For example, to create a scan that includes channels 101 to 109 of slot 1, send:

```
ROUT:SCAN:CRE (@101:109)
```

To create a scan that includes individual channels in any order, send the channels separated by commas. For example, to create a scan that scans channels 107, 102, and 109 in that order, send:

```
ROUT:SCAN:CRE (@107, 102, 109)
```

You can combine a range of channels and an individual channel in a command. For example, to create a scan that scans channels 101 to 109 and channels 107, 102, and 109, send:

```
ROUT:SCAN:CRE (@101:109, 107, 102, 109)
```

If you need to add additional channels, you can use the `ROUTE:SCAN:ADD` command. The new channels are added to the end of the scan list.

Create a scan using TSP commands

In TSP, you send a command to create the scan. You can include channels in any order in the command.

To include a range of commands, use a colon to separate the first and last channel in the list. For example, to create a scan that includes channels 101 to 109 of slot 1, send:

```
scan.create("101:109")
```

To create a scan that includes individual channels in any order, send the channels separated by commas. For example, to create a scan that scans channels 107, 102, and 109 in that order, send:

```
scan.create("107, 102, 109")
```

You can combine ranges and individual channels in commands. For example, to create a scan that scans channels 101 to 109 and channels 107, 102, and 109, send:

```
scan.create("101:109, 107, 102, 109")
```

If you need to add additional channels, you can use the `scan.add()` command. The new channels are added to the end of the scan list.

Set scan options

You can set the number of times a scan repeats, where measurements are stored, how often scans occur, whether the scan restarts if a power interruption occurs, and limits for scan measurements.

The options described here are available on the Scan tab on the SCAN menu. They are also available through remote commands.

Working with scan groups (front panel only)

When you create a scan from the front panel, sequential channels with the same functions and settings are grouped to simplify working with them. Groups allow you to make changes that apply to all channels in the group. If you add a channel with a different function to a group or change the settings for a channel, a new group is created.

Groups are always ordered sequentially.

The menu icon to the right of the scan function for the group provides options for working with groups. You can:

- **Insert Channels:** Add a channel or group of channels immediately before the selected group. If you select multiple channels with different functions, you are prompted to preserve the existing functions. If you select No, you are prompted to select a new function for the channels and the channels are maintained as a group. If you select Yes, the channels are split into multiple groups.
- **Change Channels:** Allows you to change the channels that are in the group. If you select channels that are assigned to a different function than the other channels in the group, the function is changed to match the group. If you select nonconsecutive channels, a new group with the same function is created.
- **Delete Channel or Delete Group:** Removes the selected channel or group from the scan.
- **Disable Channel or Disable Group:** Allows you to skip a channel or group during a scan while maintaining the settings. The channel or group is grayed out if it is not available. Select **Enable Channel** or **Enable Group** to use the channel or group in the scan again.
- **Copy Settings:** Copies the functions and function settings of a group.
- **Paste Settings:** Pastes copied functions and function settings to another group. If the channels in the group are sequential with another group with the same settings, the groups are merged.

The **+** below the groups adds a new group after the last group in the scan list. When you select this option, a list of channels is displayed. Select the channels you want to include in the scan. If channels have different functions assigned to them, you are prompted to preserve the existing functions. Select:

- **Yes:** Keep the existing function settings and create groups for each consecutive set of channels.
- **No:** Replace the function settings for the selected channels with one function setting. You are prompted for the new function settings.

NOTE

If the scan group was created using a configuration list, you cannot change the settings for the channels through the Settings tab.

To display the channels individually instead of in groups, select the menu icon in the upper left and select **Expand Groups**. To regroup the channels, select **Collapse Groups**.

Repeat, interval, and delay options

To repeat a scan, you can set the Scan Count. The scan repeats the number of times specified. You can also set the scan to repeat until the scan is aborted.

If your scan count is more than one, you can also set the Scan to Scan Interval. This interval is the minimum amount of time between scans. For example, if you want a scan to occur once an hour, one scan would complete normally, and the next scan occurs one hour after the first one started. When setting up intervals, be aware of measurement and other delays that could exceed the time of the interval. If other delays exceed the time of the interval, the interval is ignored, and the next scan starts immediately at completion of the previous scan.

Measurement and channel delays, measurement intervals, and other factors can affect the amount of time needed by the scan. Make sure the scan can complete during the scan interval time. If the scan interval time passes while the scan is running, the next scan starts immediately on completion of the previous scan. The Scan Duration listed on the Scan tab displays the time the scan is expected to take. Scan duration is calculated using the scan-to-scan interval setting and the channel delays.

If you are using SCPI commands, the commands for Scan Count and Scan to Scan Interval are:

- [:ROUT:SCAN:COUNT:SCAN](#) (on page 12-91)
- [:ROUT:SCAN:INTerval](#) (on page 12-97)

For example, to set a scan count of 5 with an interval of 60 seconds, send:

```
:ROUT:SCAN:COUN:SCAN 5  
:ROUT:SCAN:INT 60
```

If you are using TSP commands, the commands are:

- [scan.scancount](#) (on page 14-399)
- [scan.scaninterval](#) (on page 14-400)

For example, to set a scan count of 5 with an interval of 60 seconds, send:

```
scan.scancount = 5  
scan.scaninterval = 60
```

Specifying a reading buffer

Data from the scan is stored in a reading buffer. You can set the buffer the is used for the scan. You can also send data to a `.csv` file during the scan.

If you are specifying the buffer using remote commands and you want to use a buffer other than `defbuffer1` or `defbuffer2`, you must create the new buffer first. For information on creating a buffer, refer to [Creating buffers](#) (on page 6-6).

To specify a reading buffer from the front panel:

Select or create a buffer from the Buffer list. For information on the options when creating a buffer, refer to [Using the front panel to create reading buffers](#) (on page 6-7). If you do not make a selection, data is stored in `defbuffer1`.

To specify the buffer using SCPI commands:

Send the [:ROUTe:SCAN:BUFFer](#) (on page 12-87) command. For example, to set the scan buffer to a new buffer named `ScanData` that stores a maximum of 100,000 readings, send:

```
TRAC:MAKE "ScanData", 100000
ROUT:SCAN:BUFF "ScanData"
```

To specify the buffer using TSP commands:

Send the [scan.buffer](#) (on page 14-379) command. For example, to set the scan buffer to a new buffer named `ScanData` that stores a maximum of 100,000 readings, send:

```
ScanData = buffer.make(1000000)
scan.buffer = "ScanData"
```


Exporting data

You can send data from a scan to a `.csv` file during the scan. The `.csv` file is stored on a USB flash drive inserted in the USB port on the front panel, so you must insert and leave a USB flash drive in the USB port during the scan.

When the instrument is saving data, processing stops, so options that save data frequently slow down scan times.

Export files are limited to 500 MB. When data exceeds 500 MB, another file is created with `_n` added to the file name, where `n` starts at 1 and is incremented for each additional file.

You can send export data:

- At completion of each scan step (only available if you use remote commands)
- After each scan
- Once at the end of all scans

The options that are available when you export data are shown in the following table. These settings are used for any reading buffer data that is saved until the instrument is rebooted.

Setting	Description
Channel	Include the channel information (number and label) in a column in the saved file.
Extra Value	Available for buffers set to the Full or Writable style. Include extra values, such as the ratio component of a DCV ratio measurement, in the saved file.
File Layout	Determines how data is placed in Microsoft® Excel®: ▪ Reading per Row: Readings are displayed in rows. ▪ Reading in Channel Columns: Ignore other columns and use a special format with a column per channel. ▪ Spreadsheet Graph: Ignore other columns and use a special format that is easy to graph in Microsoft Excel.
Filename	By default, the file name is the same as the buffer name, followed by the date and time when the file was saved. The date and time are in the format <code>mmdd_hhmmss</code> . To save the file with a different name, select Change . The date and time are not included if you change the file name.
Reading Format	Include the unit, range digits, and display digit settings in the saved file.
Status	Include math, limits, and terminal settings in the saved file.

Time Format	Sets the time format: ■ Absolute: Each timestamp provides the time and date that the reading was made or the number of seconds from the first buffer reading that the reading was made. ■ None: No timestamp. ■ Parts: Timestamps contain dates, hours, minutes, seconds, and fractions of seconds according to Coordinated Universal Time (UTC). ■ Raw: Timestamps display the absolute time in seconds. ■ Relative: Timestamps are oriented to a timer with the first buffer reading timestamped at 0.000000 seconds. Each following timestamp is then based on the presently selected format.

Some export options are only available if the reading buffer is set to the style Full. Refer to [Specifying a reading buffer](#) (on page 5-38) for information on changing the buffer that is used with the scan.

To set the options for exporting scan data using SCPI commands, refer to [:ROUTe:SCAN:EXPort](#) (on page 12-95).

To set the options for export scan data using TSP commands, refer to [scan.export\(\)](#) (on page 14-386).

Restart a scan if power fails

You can set a scan to automatically restart if the scan is interrupted because power to the instrument is turned off.

If the restart option is enabled, the scan settings are saved in memory immediately after the scan is triggered and before the scan operation begins. All scan settings, including watched channels, need to be set before the scan starts. Any changes that are made after the scan starts are not recalled if power is lost and the scan needs to restart.

If the restart option is enabled and power is lost, the scan restarts when power is restored. The scan setup that was in place when the scan started becomes the power-up setup. It takes precedence over any other power-up setup. If the scan completes successfully, the scan setup is removed as the power-up setup.

If the DAQ6510 detects that a card was changed during the power-up sequence, restart is set to off, the interrupted scan is not resumed, and an event is generated. The instrument starts up normally.

When a scan is automatically restarted, it is logged in the event log.

NOTE

If a restart occurs, the original reading buffer data is retained (per the settings of the reading buffer) and a new reading buffer is created.

To enable restart from the front panel:

On the Scan tab of the Scan menu, set Power Loss Restart to **On**.

To enable restart using SCPI commands, send:

```
ROUT:SCAN:REST ON
```

To enable restart using TSP commands, send:

```
scan.restart = scan.ON
```

Setting an alarm limit for scans

You can set limits for each channel in a scan. If you set limits, a pass or fail indication is shown on the home screen when the scan runs. You can also set up an audible indicator.

You can use the Auto Learn feature to have the DAQ6510 calculate limits based on the present configuration of the system.

You can also set limits manually.

NOTE

Auto Learn sets the values for Limit 1 for the selected channels. If you manually set limits for Limit 1, the limits are overwritten by the Auto Learn values. Conversely, if you run Auto Learn and then set Limit 1 values manually, the Auto Learn values are overwritten.

Using Auto Learn to set limits

Auto Learn is available from the Scan tab of the Scan menu. Auto Learn runs a scan and establishes alarm limits based on the measurements from the scan. Make sure your system is in a stable state before running Auto Learn.

Auto Learn sets limits for each channel in the scan. When you run Auto Learn, Limit 1 is enabled and any settings that were previously set are replaced by the new values. Auto Learn does not affect Limit 2 settings.

To use Auto Learn:

1. Set up the scan.
2. On the Scan menu, select the **Scan** tab.
3. For Alarm Limits, select **Learn**.
4. Set **Window** to the percentage of deviation from the measurement that is within the limits.
5. Set **Iterations** to the number of times the scan should run to establish the limits. The instrument averages the readings to determine the limits.
6. Select **Scan**. Auto Learn runs.

To check the settings selected by Auto Learn:

1. Select the **Settings** icon. The channels and the limits set by Auto Learn for each are displayed.
2. To check the values, select a channel. You can change them if needed. The options are:
 - **Low Value:** Values below this limit display the limit failure indicator.
 - **High Value:** Values above this limit display the limit failure indicator.
 - **Enable:** Turn alarm limit checking on or off.
 - **Audible:** Determines if a beeper sounds when a measurement fails or passes.
3. Start the scan. **L1PASS** is displayed if the measurement is in the limits; **L1FAIL** is displayed if the measurement is not in the limits.

Setting manual limits for scans using the front panel

You set manual limits through the Settings tab of the Scan menu.

You can use the following procedure to adjust the settings established by Auto Learn.

To set manual limits using the front panel:

1. On the Scan menu, select the **Settings** tab.
2. Select the channels for which you want to set limits.
3. To enable limit testing, select **On**.
4. Select the **Settings** icon.
5. Set **Auto Clear** as needed. When auto clear is set to on, limit conditions are cleared automatically after each measurement. The scan will show the limit test result of the last measurement of the scan. If auto clear is off, a failed indication is not cleared until a clear command is sent.
6. Set the **Low Value**. If a measurement on the channel is below the Low Value, the limit failure indicator is displayed.
7. Set the **High Value**. If a measurement on the channel is above the High Value, the limit failure indicator is displayed.
8. The **Audible** setting determines if a beeper sounds when a measurement passes or fails. Set as needed.
9. Start the scan. **L1PASS** is displayed if the measurement is in the limits; **L1FAIL** is displayed if the measurement is not in the limits.

Setting limits for scans using SCPI

You can use the `:ROUTe:SCAN:LEARn:LIMits` and `:CALCulate2:<function>:LIMit<Y>` commands to automatically generate and set the limits. Refer to the [CALCulate subsystem](#) (on page 12-16) for command descriptions.

An example of setting limits for scans using SCPI commands:

<code>*RST</code>	Reset the instrument.
<code>:SENS:FUNC "VOLT", (@101:105)</code>	Set channels 101 to 105 to measure DC voltage with a range of 10 V and an NPLC of 0.1.
<code>:SENS:VOLT:RANG 10, (@101:105)</code>	Create a scan on channels 101 to 105.
<code>:SENS:VOLT:NPLC 0.1, (@101:105)</code>	Start the learn limits calculation, with a 1% window and 5 iterations.
<code>:ROUT:SCAN:CRE (@101:105)</code>	Query the low and high values that were set by Auto Learn.
<code>:ROUT:SCAN:LEAR:LIM 1, 5</code>	Adjust the upper limit to be 0.5 V.
<code>:CALC2:VOLT:LIM1:LOW?</code>	Start the scan.
<code>:CALC2:VOLT:LIM1:UPP?</code>	Wait for completion.
<code>:CALC2:VOLT:LIM1:UPP .5, (@101:105)</code>	Read the scan results.
<code>INIT</code>	Check to see if the readings were outside the limits.
<code>*WAI</code>	
<code>:READ? (@101:105)</code>	
<code>:CALC2:VOLT:LIM1:FAIL? (@101:105)</code>	

Setting limits for scans using TSP

You can use [scan.learnlimits\(\)](#) (on page 14-388) and [Limit channel.setdmm options](#) (on page 14-97) to automatically generate and adjust the limits. After running Auto Learn, you can use the limit options for the `channel.setdmm` command to refine the limit values.

An example of setting limits using TSP commands:

```
-- Reset the instrument.
reset()

-- Set channels 1 to 5 to measure voltage with an NPLC of 0.1 and range of 10 V.
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE,
    dmm.ATTR_MEAS_NPLC, 0.1,
    dmm.ATTR_MEAS_RANGE, 10)

-- Create a scan on channels 101 to 105.
scan.create("101:105")

-- Start the learn limits calculation, with a 1% window and 5 iterations.
scan.learnlimits(1, 5)

-- Output the values that Auto Learn set for limit 1 high and low.
print(channel.getdmm("101:105", dmm.ATTR_MEAS_LIMIT_HIGH_1))
print(channel.getdmm("101:105", dmm.ATTR_MEAS_LIMIT_LOW_1))

-- Adjust the upper limit to be 0.5 V.
channel.setdmm("101:105", dmm.ATTR_MEAS_LIMIT_HIGH_1, 0.5)

-- Start the scan and wait for completion.
trigger.model.initiate()
waitcomplete()

-- Read the scan results.
-- Check to see if the readings were outside the limits.
print(channel.getdmm("101:105", dmm.ATTR_MEAS_LIMIT_FAIL_1))
```

Scan menu options

The menu in the upper left corner of the scan screen allows you to create a new scan list, reset the scan settings, reset system settings, save the system, and expand a group.

To remove the existing steps from the scan and set up a new list, select Create New List. Your selections in the Settings, Scan, and Trigger tabs are not changed.

To clear the scan list and reset selections in the Settings, Scan, and Trigger tabs, select Reset Scan Settings. Measure settings are not affected.

To reset the instrument, select Reset System. This opens the System Info and Management screen, where you can select System Reset. This resets many of the instrument settings to their default values.

To save the system settings, including the scan settings, select Save System. This opens the Save Setup screen, where you can save most of the present settings of the instrument into a script.

To expand a group so that you can work with each channel individually, select Expand Group. To combine multiple channels into a single item, select Collapse Group. Refer to [Working with scan groups \(front panel only\)](#) (on page 5-35) for more options when working with groups.

Triggering in scans

The options on the Trigger tab of the Scan menu allow you to determine which, if any, triggers start the scan, channel action, or measurement. You can use any of the available triggers, including TRIGGER key press, external inputs, and outputs.

In addition to the standard trigger options, you can start the scan after a measurement reaches a specific value. To use this option, select the Monitor Measurement option and define the channel and limit settings. When the defined event or measurement occurs, the scan starts. For example, you could set Scan Start to Monitor Measurement and set the scan to wait until the device connected to channel 102 is above 50 °C. When channel 102 reaches 50 °C, the scan starts. The monitored channel does not need to be in the scan. The scan start values are stored in defbuffer2.

You can also send trigger notifications when a scan, channel, or measurement action completes. When you set a trigger notification for scan complete, a trigger is sent at the completion of each scan in the set of scans.

For detail on the types of triggers that are available, refer to [Triggering](#) (on page 8-18).

To set the trigger settings:

1. On the Scan menu, select the **Trigger** tab.
2. To select a trigger to start the scan, select a **Scan Start** option.
3. To select a trigger to start channel actions, select a **Start** option in the Channel section.
4. If you selected a channel trigger, you can bypass it during the first scan by selecting **Bypass Once**.
5. To send a notification when the channel action is complete, select a **Ready** option in the Channel section.
6. To select a trigger to start measure actions, select a **Start** option in the Measure section.
7. To send a notification when the measure action is complete, select a **Complete** option in the Measure section.
8. To send a notification when the scan is completed, select a **Scan Complete** option.

Run a scan

Once the scan is set up, you can run it.

To start the scan from the front panel, do one of the following:

- Press the **TRIGGER** key.
- On the SCAN menu, select **Start**.
- On the SCAN swipe screen, select **Start Scan**.
- From the measurement method indicator on the home screen, select **Initiate Scan**.

You can pause the scan from the SCAN swipe screen on the front panel. Select **Pause Scan**. To continue the scan, select **Resume Scan**.

Stop a scan

You can stop a running scan.

When you stop a scan, the channels remain in the state they were in (opened or closed) when the scan was stopped.

To stop a scan from the front panel:

- On the SCAN menu, select **Abort Scan**.
- On the SCAN swipe screen, select **Abort Scan**.
- From the measurement method indicator on the home screen, select **Abort Scan**.

Check the state of a scan

You can check the state of the scan.

Scans are based on the trigger model, so the scan states are reported as trigger model states. The trigger model states are:

- **Idle:** The trigger model is stopped
- **Running:** The trigger model is running
- **Waiting:** The trigger model has been in the same wait block for more than 100 ms
- **Empty:** The trigger model is selected, but no blocks are defined
- **Paused:** The trigger model is paused
- **Building:** Blocks have been added
- **Failed:** The trigger model is stopped because of an error
- **Aborting:** The trigger model is stopping
- **Aborted:** The trigger model is stopped
- **Success:** The scan completed successfully

While the scan is running, you can check the scan count to see how many times the scan has been repeated. The scan count increments after the scan begins. If an instrument is waiting for an input to trigger a scan start, the scan count represents the previous number of scan iterations. If the scan has yet to begin, the scan count is zero.

Each channel is added as a step to the scan. While the scan is running, you can check the step progress. This number does not increment until after the channel action completes. If the instrument is waiting for an input to trigger a channel action, the step count represents the previous step. If a step has yet to complete, the step count is zero.

To check the status from the front panel:

The SCAN swipe screen indicates if there are any overflows or values outside the limits.

The measurement method indicator at the top of the home screen indicates the status of the scan.

To check the status using SCPI commands, send:

```
:ROUTE:SCAN:STATE?
```

To check the status using TSP commands, send:

```
scanState, scanCount, stepCount = scan.state()  
print(scanState)
```

Using the SCAN swipe screen

NOTE

The SCAN swipe screen is displayed when the TERMINALS switch is set to REAR.

The SCAN swipe screen gives you front-panel access to scan operation and setup options.

If no scan has been set up, **Build Scan** opens the Scan screen, where you can create a new scan.

Figure 104: SCAN swipe screen with Build Scan displayed



Figure 105: SCAN screen with select channels dialog box displayed

The Scan Progress bar in the swipe screen displays the channels that have been scanned with the measurement that was made for the channel. Only measurements for channels selected by Watch Channels are displayed. If you have limits set, measurements that are outside the limits are shown in yellow. Overflow readings are shown in red. Measurements scroll across the screen as they are made. You cannot view earlier measurements using this screen. To view all measurements, see [Viewing and saving buffer content](#) (on page 6-21).

While the scan is running, **Remaining** displays the time before the scan completes. **Scan Count** displays the count of the scan that is running and the total count.

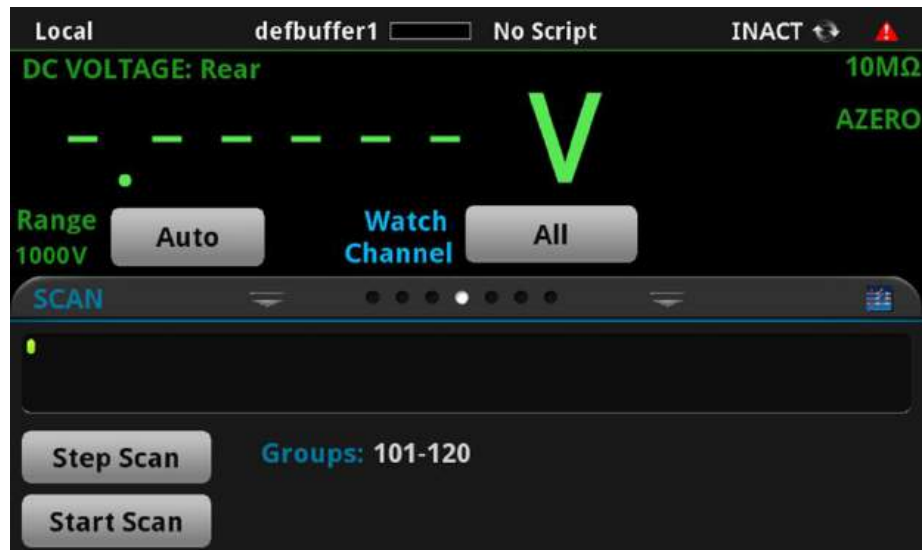
Step Scan steps through the scan step-by-step.

Start Scan starts the scan. The Start Scan button changes to Abort Scan once selected.

Abort Scan stops the scan.

Pause Scan pauses a running scan or trigger model.

Resume Scan continues a paused scan or trigger model.

Figure 106: SCAN swipe screen with Start and Step Scan options

When the scan is not running, you can edit the scan or save the scan data to a USB flash drive.

Edit opens the Scan screen, where you can change the scan settings. The icon on the right side of the swipe screen heading bar is also a shortcut to the SCAN SETTINGS menu.

Save to USB saves the scan data to a `.CSV` file on the USB flash drive.

NOTE

If you are saving large amounts of data rapidly, USB saves can stutter. To prevent this problem, set the Interval Between Scans to a time that is long enough to cover the scan duration plus the USB write time.

Figure 107: SCAN swipe screen with Edit option displayed



Remote commands for scans

Add channels to the scan list:

- SCPI: [:ROUTe:SCAN:ADD](#) (on page 12-82)
- TSP: [scan.add\(\)](#) (on page 14-374)

Add a single channel to the scan list:

- SCPI: [:ROUTe:SCAN:ADD:SINGLE](#) (on page 12-84)
- TSP: [scan.addsinglestep\(\)](#) (on page 14-376)

Write a specified value to a channel at the added step in the scan:

- SCPI: [:ROUTe:SCAN:ADD:WRITE](#) (on page 12-85)
- TSP: [scan.addwrite\(\)](#) (on page 14-377)

Define which buffer is used with the scan:

- SCPI: [:ROUTe:SCAN:BUFFER](#) (on page 12-87)
- TSP: [scan.buffer](#) (on page 14-379)

Set the first channel of the scan to wait for the channel stimulus event to be satisfied before closing:

- SCPI: [:ROUTe:SCAN:BYPass](#) (on page 12-88)
- TSP: [scan.bypass](#) (on page 14-380)

Set a trigger event that causes the channel action to occur:

- SCPI: [:ROUTe:SCAN:CHANnel:STIMulus](#) (on page 12-89)
- TSP: [scan.channel.stimulus](#) (on page 14-381)

Set the number of times the scan is repeated:

- SCPI: [:ROUTe:SCAN:COUNt:SCAN](#) (on page 12-91)
- TSP: [scan.scancount](#) (on page 14-399)

Read the number of steps in the present scan:

- SCPI: [:ROUTe:SCAN:COUNt:STEP?](#) (on page 12-92)
- TSP: [scan.stepcount](#) (on page 14-403)

Delete the existing scan list and create a new list of channels to scan:

- SCPI: [:ROUTe:SCAN:CREate](#) (on page 12-93)
- TSP: [scan.create\(\)](#) (on page 14-383)

Store data from a scan to a file on a USB flash drive:

- SCPI: [:ROUTe:SCAN:EXPort](#) (on page 12-95)
- TSP: [scan.export\(\)](#) (on page 14-386)

Set the interval time between scan starts when the scan count is more than one:

- SCPI: [:ROUTe:SCAN:INTerval](#) (on page 12-97)
- TSP: [scan.scaninterval](#) (on page 14-400)

Automatically calculate alarm limits based on the present configuration of the system:

- SCPI: [:ROUTe:SCAN:LEARn:LIMits](#) (on page 12-98)
- TSP: [scan.learnlimits\(\)](#) (on page 14-388)

Return a list that includes the initial open or close state of any cards installed in the instrument and the settings at each step of the scan (TSP only):

- TSP: [scan.list\(\)](#) (on page 14-389)

Select the trigger that starts the measurement:

- SCPI: [:ROUTe:SCAN:MEASure:STIMulus](#) (on page 12-99)
- TSP: [scan.measure.stimulus](#) (on page 14-391)

Set the interval time between measurements:

- SCPI: [:ROUTe:SCAN:MEASure:INTerval](#) (on page 12-102)
- TSP: [scan.measure.interval](#) (on page 14-390)

Set the relay action when the scan starts:

- SCPI: [:ROUTe:SCAN:MODE](#) (on page 12-103)
- TSP: [scan.mode](#) (on page 14-393)

Specify the channel to monitor for a limit to be reached before starting the scan:

- SCPI: [:ROUTe:SCAN:MONitor:CHANnel](#) (on page 12-104)
- TSP: [scan.monitor.channel](#) (on page 14-394)

Set the low limit to be used by the scan monitor:

- SCPI: [:ROUTe:SCAN:MONitor:LIMit:LOWer\[:DATA\]](#) (on page 12-105)

- TSP: [scan.monitor.limit.low.value](#) (on page 14-396)

Set the high limit to be used by the scan monitor:

- SCPI: [:ROUTe:SCAN:MONitor:LIMit:UPPer\[:DATA\]](#) (on page 12-107)
- TSP: [scan.monitor.limit.high.value](#) (on page 14-395)

Set whether a scan starts immediately when triggered or after measurements reach a set value:

- SCPI: [:ROUTe:SCAN:MONitor:MODE](#) (on page 12-106)
- TSP: [scan.monitor.mode](#) (on page 14-397)

Set a scan to automatically restart if it was interrupted by a power failure:

- SCPI: [:ROUTe:SCAN:REStart](#) (on page 12-108)
- TSP: [scan.restart](#) (on page 14-398)

Set the event that starts the scan:

- SCPI: [:ROUTe:SCAN:StARt:STIMulus](#) (on page 12-109)
- TSP: [scan.start.stimulus](#) (on page 14-401)

Run the scan:

- SCPI: [:INITiate\[:IMMediate\]](#) (on page 12-60)
- TSP: [trigger.model.initiate\(\)](#) (on page 14-478)

Stop the scan:

- SCPI: [:ABORt](#) (on page 12-60)
- TSP: [trigger.model.abort\(\)](#) (on page 14-476)

Check the state of the scan:

- SCPI: [:ROUTe:SCAN:StARt?\[:STATe?\]](#) (on page 12-111)
- TSP: [scan.state\(\)](#) (on page 14-400)

Multiple-channel operation

Normally, when you close an input channel, other channels on the switching module close automatically to internally connect it to the DMM of the instrument. Using multiple-channel operation, you can also control each switching module channel independently.

Multiple-channel operation allows any channels in the test system to be closed or opened at the same time. It also allows independent control of nonmeasurement channels, such as backplane isolation channels and paired channels. Multiple-channel operation should only be performed by experienced test system engineers.

Multiple-channel operation provides you with added flexibility in how you use a switching module. For example, assume you want to route a signal into channel 1 and out channel 20 of a 7700 switching module. You would do this by closing channels 1, 20, and 23. If you open channels 24 and 25, you isolate the input signal from the DMM of DAQ6510.

For a multiplexer card such as the 7701, which is normally a 1 x 32 multiplexer, using multiple-channel operation to control channels independently allows you to configure the cards as two independent 1 x 16 multiplexers. For a card such as the 7707, which is normally used as a single 1 x 10 multiplexer, controlling channels independently allows you to configure the 7707 as two 1 x 5 multiplexers.

For the 7710, channels 21, 22, and 23 are configured automatically by the DAQ6510 when using normal channel operation. However, by using multiple-channel operation, you can manually configure them. For example, assume a 7710 module is installed in slot 1 and an external source is connected to the Card Source input. To connect the source to channel 21 only, use multiple-channel operation.

For the 7711 and 7712, multiple-channel operation provides control of one or two switching modules. OUT A automatically switches as appropriate for channels 1 to 4, and OUT B automatically switches as appropriate for channels 5 to 8 for each module. When you close two channels in a single module (one for OUT A and one for OUT B), the appropriate channels will close channels outside of the banks and are not affected by channel closures.

When you close a channel using multiple-channel operation, the specified channels are closed without affecting any other channels, including paired channels. The action of the close command depends on which, if any, function is set for the DMM.

If no function is set, the listed channels or channel pairs are closed. You can select multiple channels.

When you close a channel or channel pair, the instrument:

- Closes the items in the list of channels.

- Incurs the settling time and any user-specified delay.

Multiple-channel operation is not available for digital I/O, digital-to-analog converter (DAC analog output), and totalizer channels.

When multiple-channel operation is completed, it is best practice to open all channels.

WARNING

Careless multiple-channel operation could create an electric shock hazard that could result in severe injury or death. Improper operation can also cause damage to the switching modules and external circuitry. Operating channels independently should be restricted to experienced test engineers who recognize the dangers associated with multiple independent channel closures. Do not attempt to perform this procedure unless you are qualified, as described by the types of product users in the [Safety precautions](#) (on page 1-1). Do not perform these procedures unless qualified to do so. Failure to recognize and observe normal safety precautions could result in personal injury or death.

WARNING

Most switching modules use latching relays. Closed channels remain closed when the DAQ6510 is turned off. Never handle a switching module that is connected to an external source that is turned on. Turn off all power sources before making or breaking connections to the switching module and before installing or removing a switching module. Failure to disconnect all power may expose you to hazardous voltages, that, if contacted, could cause personal injury or death.

CAUTION

To prevent damage to a switching module, do not exceed the maximum signal level input for that switching module. Most switching modules are rated for 303 V. To verify the maximum switching module voltage, use the SCPI command [:SYSTem:CARD<slot>:VMAX?](#) (on page 12-214) or the TSP command [slot\[slot\].maxvoltage](#) (on page 14-419). If you are controlling channels individually, the OVERFLOW message occurs when the maximum voltage of the DAQ6510, 1010 V, is exceeded.

Multiple-channel operation using the front panel

When Channel Access is set to Full, you can use multiple-channel operation through the front panel. When Full is selected, you can control the backplane and function relays and close multiple multiplexer channels at the same time.

To control the channels, set the function of the channel to None. When a channel is set to None, the DAQ6510 no longer prompts you to set a function for that channel when it is selected for a scan or to be closed. After a system reset, the prompts to set a function are restored.

From the CHANNEL swipe screen, you can directly control the channels using the Select button. When you choose Select, use the dialog box that is displayed to close or open the channels.

You can also use the Channel Control screen to close and open the channels.

To set Channel Access:

1. Select the **MENU** key.
2. Select System **Settings**.
3. Scroll to the bottom of the screen.
4. Set Channel Access to **Full**.

This setting is saved through a power cycle.

Control multiple channels from a remote interface

You can use remote commands to control multiple-channel operation.

The SCPI commands are:

- [:ROUTe\[:CHANnel\]:MULTiple:CLOSE](#) (on page 12-74): This command closes the specified channels without affecting any other channels, including paired channels.
- [:ROUTe\[:CHANnel\]:MULTiple:OPEN](#) (on page 12-75) This command opens the channels in the channel list without affecting any others.

The TSP commands are:

- [channel.multiple.close\(\)](#) (on page 14-82)
- [channel.multiple.open\(\)](#) (on page 14-82)

For example, to disconnect channels 11 to 20 from channels 1 to 10 on a 7706 switching module, you send the SCPI command:

```
:ROUT:MULT:CLOSE (@126)
```

Send the TSP command:

```
channel.multiple.close("126")
```

Avoid corrupt measurements when operating channels independently

Improper use of multiple-channel operation can result in corrupt measurements. For example, assume two 7700 modules installed in slots 1 and 2 with a 2-wire measurement function selected. If you use close channels 201 and 225, you connect the input at channel 201 to the DAQ6510 for measurement.

If you then close channel 101, channel 125 also closes to connect the input at channel 101 to the DAQ6510. You now have two input channels (101 and 201) connected to DMM input at the same time.

You can avoid this problem by opening channels 201 and 225 before closing channels 101 and 125:

1. Close channels 201 and 225.
2. Open channels 201 and 225.
3. Close channel 101.

Dual independent multiplexers

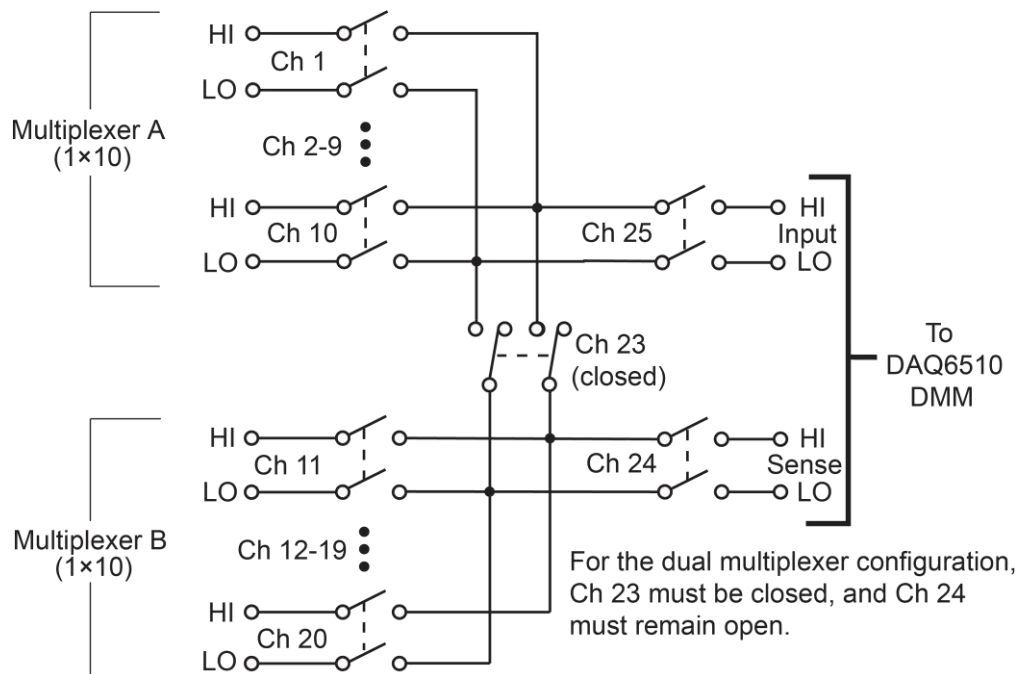
Using multiple-channel operation, you can configure any multiplexer switching module as two independent multiplexers. For example, the Model 7700 is normally used as a single 1 x 20 multiplexer, but it can also be configured as two 1 x 10 multiplexers.

To configure a multiplexer switching module as two multiplexers, use multiple-channel operation to close the 2-pole and 4-pole relays. For example, configure the 7700 as two independent multiplexers by closing channel 23. As shown in the following figure, the closed position of channel 23 isolates Multiplexer A (channels 1 through 10) from Multiplexer B (channels 11 through 20).

For the dual multiplexer configuration, only Multiplexer A channels can be internally connected to the DMM of the DAQ6510. For the 7700, closing channel 25 allows channels 1 through 10 to be measured by the DMM.

When using the dual multiplexer configuration, the sense backplane isolation relay must be kept open to isolate Multiplexer B channels from the sense terminals of the DMM. For the 7700, channel 24 must be kept open, as shown in the following figure.

Figure 108: Example 7700 dual multiplexer configuration

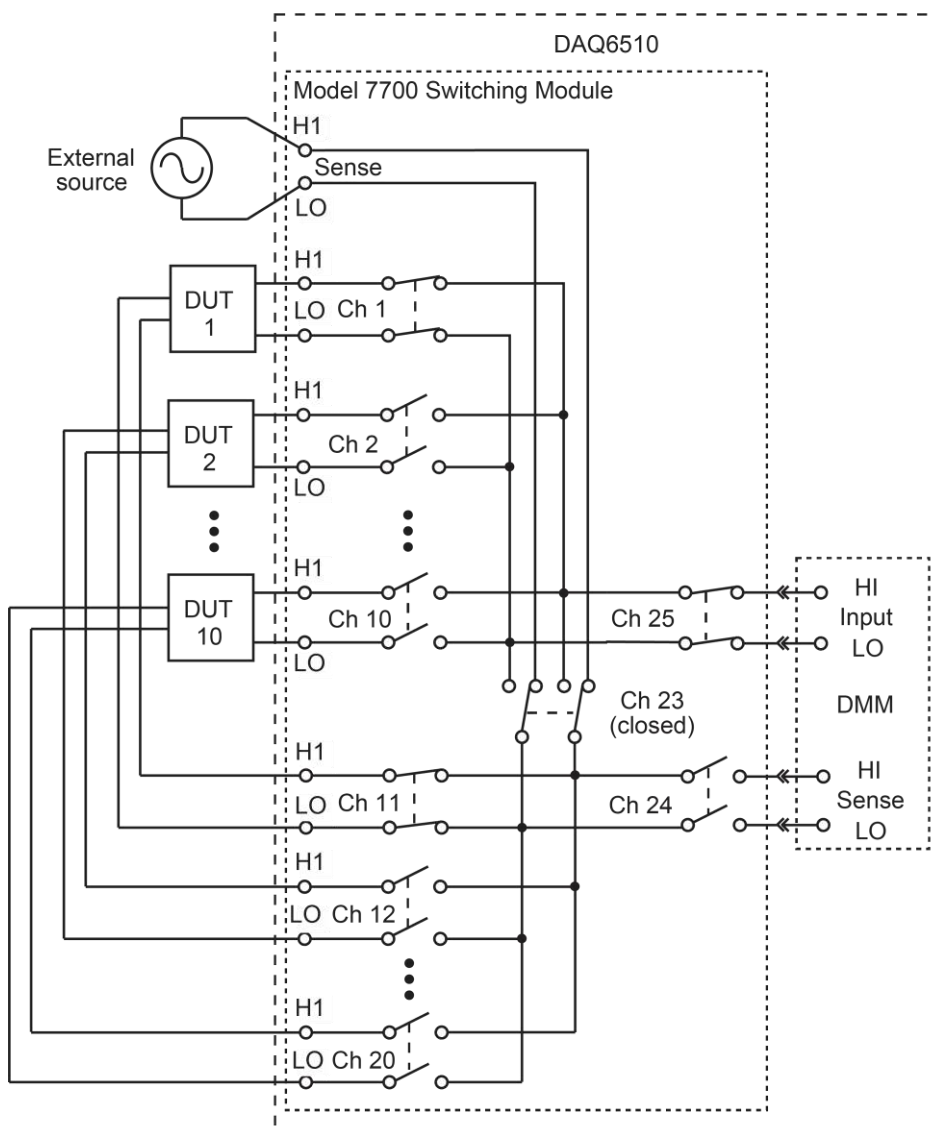


7700 dual multiplexer application

This application demonstrates how to use the 7700 20-Channel Differential Multiplexer Switching Module as a dual multiplexer to bias and measure 10 DUTs. An external source powers the DUT, while DMM of the DAQ6510 measures the DUT output. To prevent overloading of the external source, each DUT is powered and measured separately.

The following figure shows the connections for this application. The external source is connected to the sense terminals of the switching module, and DUT is connected to channels 101 through 110. Channels 111 through 120 are used to connect external power to each DUT.

Figure 109: Connections for dual multiplexer example



For this application, the 2- and 4-pole relay and backplane isolation relays of the switching module are controlled as follows:

- Closing channel 23 connects the external source to the DUT using channels 11 to 20. This also isolates the measure channels (1 to 10) from the source channels (11 to 20). This channel must remain closed while testing the DUT.
- Opening channel 24 isolates the external source from the backplane of the DAQ6510. This channel must remain open while testing the DUT.
- Closing channel 25 connects an input channel (1 to 10) to the DMM.

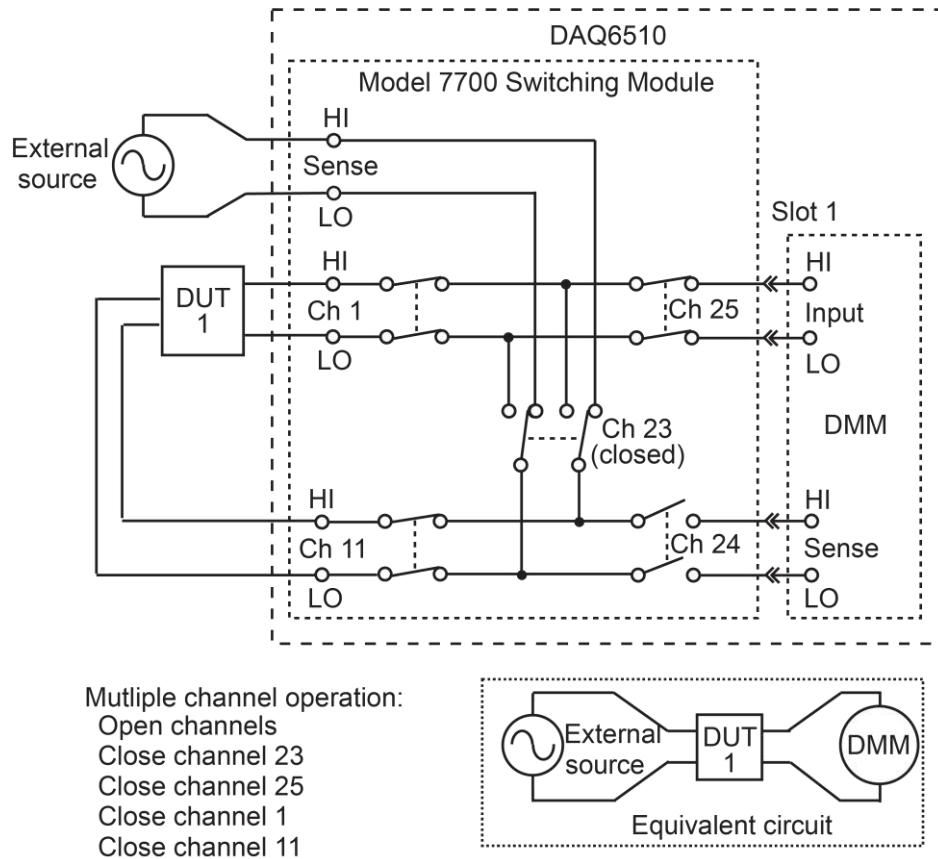
In the previous figure, channels 1 and 11 are closed to test DUT 1. A more detailed diagram of the same test is shown in the next figure. The tests for the other DUTs are similar except that different source and measure channels are closed. Closed channels for each DUT test are:

Device under test	Closed channels
DUT 1	1, 11, 23, and 25
DUT 2	2, 12, 23, and 25
DUT 3	3, 13, 23, and 25
DUT 4	4, 14, 23, and 25
DUT 5	5, 15, 123, and 25
DUT 6	6, 16, 23, and 25
DUT 7	7, 17, 23, and 25
DUT 8	8, 18, 23, and 25
DUT 9	9, 19, 23, and 25
DUT 10	10, 20, 23, and 25

The following test procedure assumes a 7700 switching module is installed in slot 1 of the mainframe.

The procedure assumes that the instrument is operating in the continuous measurement (triggering) mode.

Figure 110: Testing DUT 1



To use the 7700 as a dual multiplexer to measure multiple DUTs:

1. Set the TERMINALS switch on the front panel to **REAR**.
2. Set DAQ6510 channel access to **FULL**.
3. Open all channels.
4. Set the measurement function of channel 1 to measure **DC Voltage**.
5. Close channel 25.
6. Open channel 23.
7. Set the measurement function of channel 11 to **None**.
8. Modify the steps based on the previous table to test the second DUT.
9. Test the remaining eight DUTs following the same procedure.
10. After the last DUT is tested, repeat step 1 to open all channels.

Reading buffers

In this section:

Introduction to reading buffers.....	6-1
Getting started with buffers	6-2
Remote buffer operation	6-30
Apply mathematical expressions to reading buffer data.....	6-40
Using buffers across TSP-Link nodes	6-43

Introduction to reading buffers

Reading buffers capture measurements, ranges, data collected from scans, and instrument status. The DAQ6510 has two default reading buffers. You can also create user-defined reading buffers.

Reading buffers provide statistics, including average, minimum, maximum, and standard deviation. If you use SCPI commands over the remote interface, peak-to-peak statistics are also available.

When you create a reading buffer, that buffer becomes the active buffer until you choose a different buffer.

You can perform the following operations on reading buffers from the front panel or a remote interface:

- Configure, store, and recall reading buffers.
- View reading buffer content.
- Choose to store readings in a default reading buffer or the user-defined reading buffers.
- Save reading buffer content to a USB flash drive.
- Set reading buffers to fill once or fill continuously.
- Change the capacity of reading buffers.
- Delete user-defined reading buffers. You cannot delete `defbuffer1` and `defbuffer2`.
- Clear reading buffers.
- Clear the default reading buffers and delete the user-defined reading buffers by turning the instrument off or sending an instrument reset command.

Getting started with buffers

The following sections provide you with information to help you start using reading buffers. The [Remote buffer operation](#) (on page 6-30) section provides additional information about accessing the reading buffers with remote commands.

Types of reading buffers

There are two default buffers, `defbuffer1` and `defbuffer2`.

If you do not select a specific buffer, all readings are stored in `defbuffer1`. If you want to store readings in `defbuffer2`, you need to select it.

If you want to store readings in a user-defined buffer, you need to create the buffer. The user-defined buffer is automatically set to be the active buffer. New readings are stored in the active buffer.

For information about writable reading buffers, see [Writable reading buffers](#) (on page 6-36).

Effects of reset, scan start, and power cycle on buffers

The instrument clears the default buffers when a reset command is sent or when the power is turned off and then turned on again.

The instrument deletes all user-defined buffers when a reset command is sent or when the power is turned off and then turned on again.

The active buffer is cleared when the function is changed using the front panel. The buffer that is set to collect data for a scan is cleared when a scan is initiated.

Buffer fill status

There are several ways to view buffer fill status from the front panel.

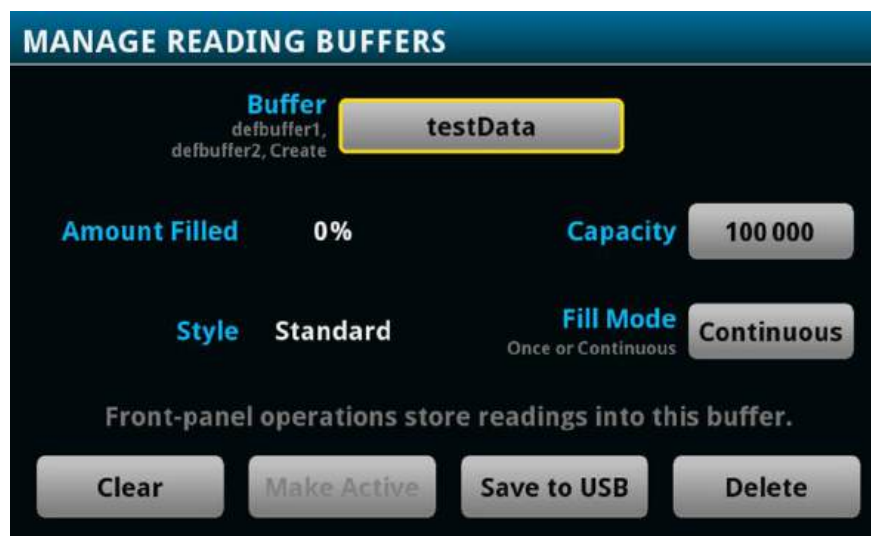
As shown in the following figure, the active buffer indicator on the home screen displays buffer fill status and the [STATISTICS swipe screen](#) (on page 3-22) displays buffer statistics. Refer to [Active buffer indicator](#) (on page 3-12) for more information on the indicator. To view the STATISTICS swipe screen, swipe the bottom of the screen.

Figure 111: STATISTICS swipe screen



The MANAGE READING BUFFERS window displays buffer fill status as the Amount Filled, as shown in the following figure.

Figure 112: MANAGE READING BUFFERS window



The System Events tab on the [System Event Log menu](#) (on page 3-67) displays the following buffer events:

- Event code 4915, "Attempting to store past the capacity of reading buffer," which occurs when a buffer that is set to fill once is full.
- Event code 4917, "The fill status of `defbuffer1` is 0% filled" or "The fill status of `defbuffer2` is 0% filled" if a default buffer is at 0% capacity.
- Event code 4918, "Reading buffer `defbuffer1` is 100% filled" or "Reading buffer `defbuffer2` is 100% filled" if a default buffer is at 100% capacity.

Figure 113: System Events tab

System Events		Log Settings	
Time	Code		Description
04/13 17:27	4917		Reading buffer <code>defbuffer1</code> is 0% filled
04/13 17:27	4917		Reading buffer <code>defbuffer2</code> is 0% filled
04/13 17:51	4915		Attempting to store past capacity of reading buffer

NOTE

Reading buffer status is only reported if the log state is set to on. Refer to the SCPI command description [:TRACe:LOG:STATE](#) (on page 12-241) or TSP command description [bufferVar.logstate](#) (on page 14-54) for detail on setting the log state for a buffer.

Timestamps

The measurements in the reading buffers contain timestamps. Readings start at the first entry in the empty reading buffer. Readings are then taken sequentially until the end of the buffer is reached. If the buffer fill mode is continuous, readings wrap to the first entry and overwrite the older data. The relative time is taken from the first reading made after a buffer is cleared.

For a buffer that fills once, the first entry starts at index 1 with the timestamp in absolute time. For continuous buffers, the lowest timestamp is after the last entry. For example, if you take 150 readings into a buffer with a capacity of 100, the last reading is at entry 50 and the earliest reading is at 51.

Creating buffers

To create a new user-defined reading buffer, you need to provide a name, capacity, and style for the new buffer.

User-defined buffer names must start with an alphabetic character. The names cannot contain any periods or the underscore (`_`) character. The name can be up to 31 characters long. If you create a reading buffer that has the same name as an existing user-defined buffer, the existing buffer is overwritten by the new buffer. Any data in the existing buffer is lost.

You cannot assign reading buffers the name `defbuffer1` or `defbuffer2`. In addition, the buffer name must not already exist as a global variable, a local variable, table, or array.

When you create a buffer, you set a buffer style. The buffer style controls the amount of information that is saved with each reading in the reading buffer. Buffer styles are:

- **Standard:** Store readings with full accuracy with formatting.
- **Full:** Store the same information as standard, plus additional information, such as the ratio component of a DCV ratio measurement.
- **Writable:** Manually write external data to a reading buffer. For more information, see [Writable reading buffers](#) (on page 6-36). You cannot select this buffer style from the front panel; you must use remote commands.
- **Full Writable:** Manually write external data to a reading buffer with two values per buffer index. You cannot select this buffer style from the front panel; you must use remote commands.

NOTE

You can only select the style of the reading buffer when you first create the buffer. Not all remote commands are compatible with the writable and full writable buffer styles. Check the Details section of the command descriptions before using them with any of these buffer styles.

There is no fixed limit on the number of user-defined reading buffers you can create. However, you are limited by available memory in the instrument.

When you create a reading buffer, it becomes the active buffer. If you create two reading buffers, the last one you created becomes the active buffer.

The following topics provide information about using the front panel to create buffers and introduce how to use remote commands to create buffers.

For additional information about using remote commands for buffer operations, see the following sections of this manual:

- [Remote buffer operation](#) (on page 6-30)
- SCPI commands, see [TRACe subsystem](#) (on page 12-229)
- TSP commands, see [TSP commands](#) (on page 14-10)

To use the front panel to create a reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 114: MANAGE READING BUFFERS window



3. Select **Buffer**.
4. Select **Create New**. A keyboard is displayed.
5. Enter a name for the buffer you are creating. The following figure uses the name `testData`.

Figure 115: Buffer Name keyboard



6. Select the **OK** button on the displayed keyboard.
7. The Style dialog box is displayed. You can select:
 - **Standard**: Store readings with full accuracy with formatting.
 - **Full**: Store the same information as standard, plus additional information, such as the ratio component of a DCV ratio measurement.

Figure 116: Select the buffer style



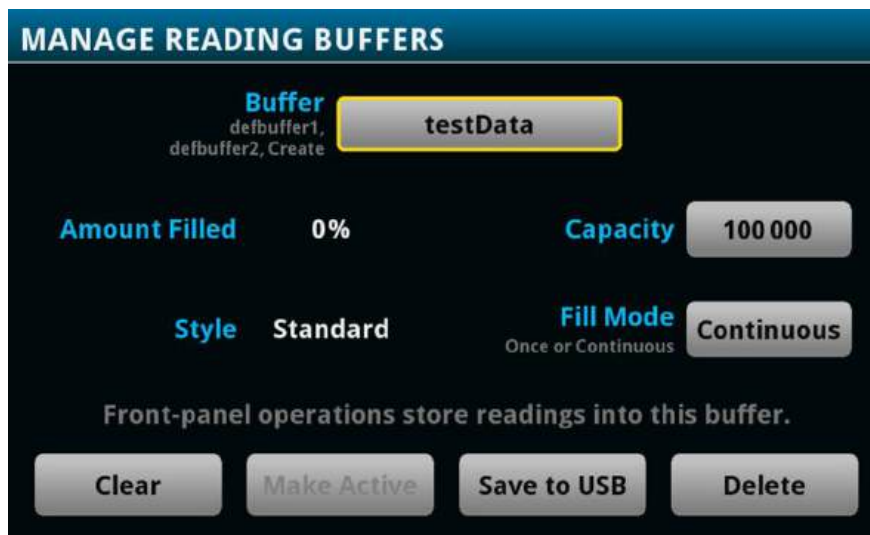
8. The Initial Capacity window is displayed. Enter the number of readings that the buffer will hold. To set the capacity to the maximum buffer size, based on the available memory of the instrument, enter **0**.

Figure 117: Initial Capacity window



9. Select **OK**. The MANAGE READING BUFFERS window is displayed, showing the buffer you just created.

Figure 118: MANAGE READING BUFFERS window



10. Press the **HOME** key to return to the home screen.

After you create a new reading buffer, the new reading buffer becomes the active buffer.

Figure 119: Active buffer indicator



NOTE

You can select the reading buffer indicator to open a menu of available buffers. Select a buffer name in the list to make it the active reading buffer. The name of the new active reading buffer is updated in the indicator bar. The green bar next to the buffer name indicates how full the buffer is. To create a new buffer, select Create New from the reading buffer indicator.

To use SCPI commands to create a reading buffer:

To create a full reading buffer named `testData` with a capacity of 200 readings, send the following command:

```
TRACe:MAKE "testData", 200, FULL
```

To use TSP commands to create a reading buffer:

To create a full reading buffer named `testData` with a capacity of 200 readings, send the following command:

```
testData = buffer.make(200, buffer.STYLE_FULL)
```

Setting reading buffer options

You can specify the settings for the reading buffers. The settings you can select include:

- **Buffer capacity:** The amount of data the buffer holds
- **Fill mode:** How the incoming data is managed as the buffer fills

Setting reading buffer capacity

The capacity of a reading buffer determines how many readings that buffer holds. You can change the capacity of reading buffers.

NOTE

Readings and statistics that are stored in the reading buffer are deleted when you change the capacity of a buffer.

For user-defined buffers, you assign a capacity when you create the reading buffer. For default buffers (`defbuffer1` and `defbuffer2`), the initial buffer size is 100,000 readings. If you specify 0 for the size of the buffer, the instrument assigns the largest buffer size possible based on the available memory.

The available capacity for reading buffers is affected by other operations of the instrument. Factors that can affect buffer capacity include:

- Other reading buffers. The total capacity of the reading buffers in the instrument affects the space that can be allocated to a new buffer.
- Scripts that are loaded onto the instrument.
- Applications that are loaded onto the instrument.
- Configuration lists.
- Scripts that are actively running.
- Variables that reside in the run-time environment.
- The TriggerFlow® trigger model.
- USB drives that contain files that use a lot of memory, such as image files.
- TSP-Link nodes.
- Scans.

CAUTION

The DAQ6510 notifies you when the system runs out of memory. If the instrument encounters memory allocation errors (errors that specifically state “Out of Memory”), the state of the instrument cannot be guaranteed. After attempting to save any important data, turn off power to the instrument and turn it back on to reset the runtime environment and return the instrument to a known state. Unsaved scripts and reading buffers will be lost.

To maximize the memory available for reading buffers:

- Delete user-defined buffers that you are not using. If you power cycle the instrument, all user-defined buffers in the instrument are deleted.
- If you are not using the default buffers, set them to the minimum capacity of 10.
- Remove the USB flash drive from the USB port.
- Reduce the number of scripts. Refer to [Working with scripts](#) (on page 13-7) for information on managing scripts.
- Reduce the number of configuration lists or the number of indices stored in configuration lists. Refer to [Configuration lists](#) (on page 4-99) for information on managing configuration lists.
- Reduce the number of TSP-Link nodes.
- Delete unneeded global variables from the runtime environment by setting them to `nil`.
- Adjust the `collectgarbage()` settings in Lua. See [Lua memory management](#) (on page 13-35) for more information.

- Review scripts to improve their memory usage. In particular, you can see memory gains by changing string concatenation lines into a Lua table of string entries. You can then use the `table.concat()` function to create the final string concatenation.

For additional information on memory use by the instrument, refer to [Memory considerations for the runtime environment](#) (on page 13-50).

The overall capacity of all buffers stored in the instrument can be up to 6,000,000 readings for standard reading buffers.

The maximum capacity for a reading buffer that is set to the Full style is less than one set to the Standard style. Full style buffers store more data for each reading.

The following topics describe how to set the reading buffer capacity.

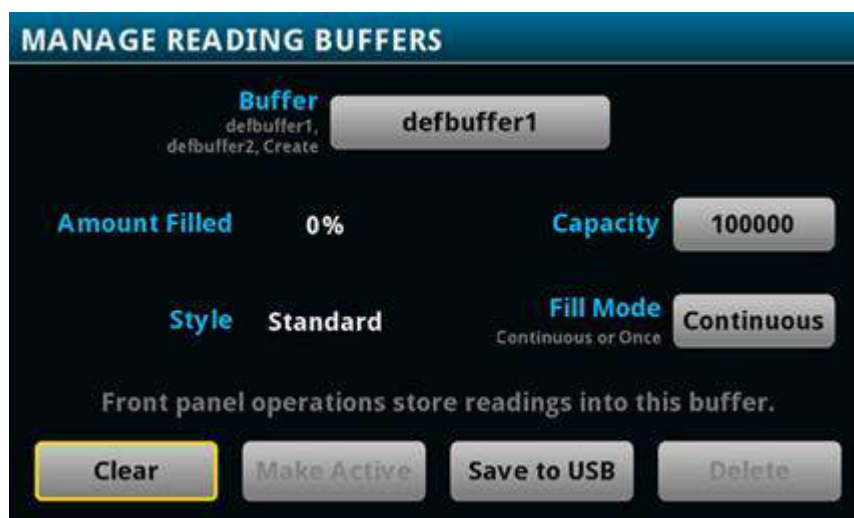
Using the front panel to set buffer capacity:

NOTE

When you resize a reading buffer, data in the buffer is cleared.

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 120: MANAGE READING BUFFERS window



3. Select a reading buffer from the list. For example, select `testData`. The settings for `testData` are displayed.
4. Select **Capacity** and enter the new size for the buffer.

Figure 121: Buffer Capacity number pad



5. Select **OK**. The MANAGE READING BUFFERS window is displayed.
6. Press the **HOME** key to return to the home screen.

Using SCPI commands to set buffer capacity:

To set the `testData` reading buffer to hold 300 readings, send the following command:

```
TRACe:POINts 300, "testData"
```

Using TSP commands to set buffer capacity:

To set the `testData` reading buffer to hold 300 readings, send the following command:

```
testData.capacity = 300
```

Setting the fill mode

The fill mode setting for the reading buffer controls how the incoming data is managed as the buffer fills. You can set the read buffer to:

- **Fill once:** The buffer stops accepting data once it fills to capacity. When the buffer reaches capacity, no more readings are made and event code 4915, "Attempting to store past capacity of reading buffer," is displayed.
- **Fill continuously:** Data fills the buffer normally until the end of the buffer is reached. When the end is reached, the data returns to the beginning of the buffer and overwrites the oldest reading. This is a traditional circular buffer. In this case, the buffer never technically fills.

NOTE

When readings are made using a high sample rate and stored into a continuous reading buffer with a capacity of less than 10,000 readings, the instrument may not be able to fully process the incoming data before it is overwritten with new data. This can result in gaps in graph traces and the loss of statistics and histogram information. To prevent these problems, increase the buffer capacity or reduce the sample rate. If you are measuring with a measure count of more than 1 into a continuous buffer, size your buffer two or more times greater than the measure count to allow graphing and other system operations to work more efficiently.

The following topics describe how to set the reading buffer fill mode.

Using the front panel to set fill mode:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 122: MANAGE READING BUFFERS window

3. Select a reading buffer from the list. For example, select `testData`. The settings for `testData` are displayed.
4. Select the **Fill Mode** option.
5. Press the **HOME** key to return to the home screen.

Using SCPI commands to set the buffer fill mode:

To set the `testData` reading buffer fill mode to continuous, send the following command:

```
TRACe:FILL:MODE CONT, "testData"
```

To set the `defbuffer1` reading buffer fill mode to fill once, send the following command:

```
TRACe:FILL:MODE ONCE, "defbuffer1"
```

To get the fill mode that is set, send the following command:

```
TRACe:FILL:MODE? "defbuffer1"
```

Where a return of `ONCE` indicates the buffer is set to fill once and a return of `CONT` indicates the buffer is set to fill continuously.

Using TSP commands to set a buffer fill mode:

To set the `testData` reading buffer fill mode to continuous, send the following command:

```
testData.fillmode = buffer.FILL_CONTINUOUS
```

To set the `defbuffer1` reading buffer fill mode to fill once, send the following command:

```
defbuffer1.fillmode = buffer.FILL_ONCE
```

To print the `defbuffer1` fill mode setting, send the following command:

```
print(defbuffer1.fillmode)
```

Where a return of 0 indicates the buffer is set to fill once and a return of 1 indicates the buffer is set to fill continuously.

Selecting a buffer

The default reading buffer is `defbuffer1`. You can also use a different buffer (`defbuffer2` or a user-defined reading buffer).

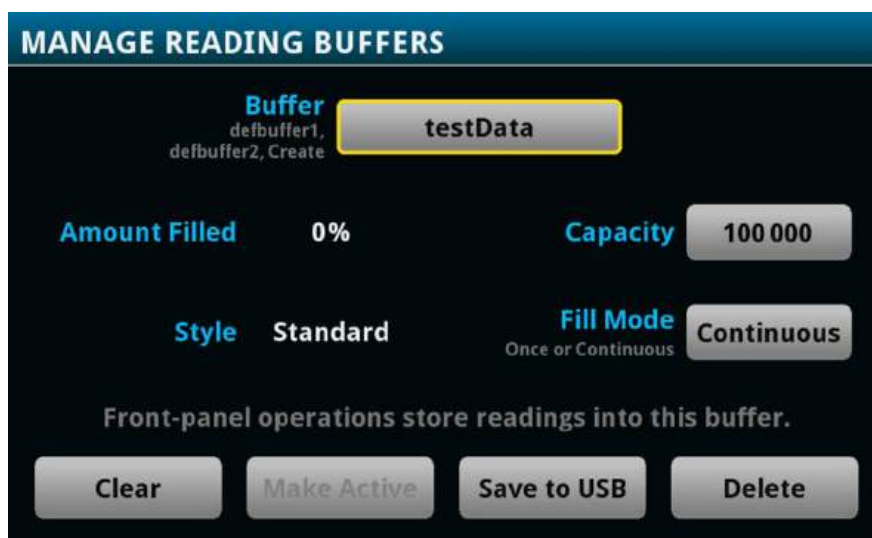
When you use remote commands to create buffers, the buffers are available to the system and can be used with any command that takes a buffer parameter. A newly created buffer automatically becomes the active buffer. If the active buffer is deleted, `defbuffer1` becomes the active buffer.

Using the front panel:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.

Figure 123: MANAGE READING BUFFERS window

3. Select a reading buffer from the list. In the example below, `testData` is selected.

Figure 124: MANAGE READING BUFFERS window

4. Select the **Make Active** button. The "Are you sure" dialog box is displayed.
5. Select **Yes**.

You can also select reading buffers from the active buffer indicator on the home screen. Refer to [Active buffer indicator](#) (on page 3-12) for information about using the indicator to select buffers.

Using SCPI commands to select a reading buffer:

To make a measurement and store the readings in a specific reading buffer, send the command:

```
:READ? "<bufferName>"
```

If you do not specify a buffer name, readings are stored in `defbuffer1`.

An alternative to sending the `:READ? "<bufferName>"` command is to send the command:

```
:TRACe:TRIGger "<bufferName>"
```

The `:TRACe:TRIGger` command stores readings in the specified reading buffer. If no buffer is specified for the parameter, `defbuffer1` is used. To see the readings stored in the buffer after using this command, use the `:FETCh?` command to see the last reading stored in the buffer or the `:TRACe:DATA?` command to see multiple readings from the buffer.

NOTE

To specify a user-defined reading buffer, you must create the buffer first.

To select current as the measurement function, measure current, and return the readings in the `testData` reading buffer, send the following commands:

```
:SENSe:FUNCTION "CURRent"  
:READ? "testData"
```

To measure current and store the readings in the `defbuffer2` reading buffer, send the following command:

```
:MEASure:CURRent? "defbuffer2"
```

To measure voltage and store the readings in the `defbuffer2` reading buffer, send the following command:

```
:MEASure:VOLTage? "defbuffer2"
```

To measure current and return the relative time and a reading, send the following command:

```
:MEASure:CURRent? "testData", REL, READ
```

Buffer storage is consistent whenever readings are taken. Parameters such as `REL` and `READ` only affect what is included in the response. If you do not include parameters, the command only returns the reading.

Using TSP commands to select a reading buffer:

To make a measurement and store the readings in a specific reading buffer, use the `dmm.measure.read(bufferName)` function. If you do not specify a buffer when you use the `dmm.measure.read()` function, readings are stored in `defbuffer1`.

To measure DC voltage and store the readings in the `voltMeasBuffer` reading buffer, send the commands:

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE  
dmm.measure.read(voltMeasBuffer)
```

To measure voltage, store the readings in `voltMeasBuffer`, and print the last reading in the buffer, send the command:

```
print(dmm.measure.read(voltMeasBuffer))
```

To measure DC current, store the readings in `defbuffer1`, and print the last reading in the buffer, send the commands:

```
dmm.measure.func = dmm.FUNC_DC_CURRENT  
print(dmm.measure.read())
```

Viewing and saving buffer content

You can view the content of buffers from the front panel.

You can also save the contents of the reading buffer to a USB flash drive. The stored file can be loaded directly into Microsoft® Excel® or another tool. The file contains all the information the instrument records about each data point in the reading buffer. When you save the buffer data, you may indicate a starting or ending point to save only a portion of the data. If you do not specify a starting and ending point, the entire buffer data is saved. You may also specify how you want the time saved with the time format parameter.

Using remote commands, you can append the contents of a reading buffer to a file that is already on the USB flash drive. When you append data, you can specify the starting and ending point in the buffer to save only a portion of the data.

All readings are saved in the comma-separated value (.csv) file format. This format stores tabular data (numbers and text) in plain-text form. You can import the .csv file into a spreadsheet.

NOTE

The header rows in the .csv file are not fixed. They may change as buffers are enhanced or when additional information is needed.

Export files are limited to 500 MB. When data exceeds 500 MB, another file is created with *n* added to the file name, where *n* starts at 1 and is incremented for each additional file.

You can view data from the reading buffers through the front panel using the Reading Table. The Reading Table displays the following information:

- **Index:** The sequential number of the reading.
- **Time:** The date and time of the reading.
- **Reading:** The data that was measured.
- **Extra:** Only displayed for buffers that are set to Full. The extra value that is stored with a reading, such as the ratio component of a DCV ratio measurement.
- **Channel:** The channel that was used to make the measurement. If the front terminals are selected, `Front` is displayed.

If you select a data point, additional detail about that data point is displayed, including the function, math, and limits.

To jump to a specific spot in the data, select the menu in the upper left and select **Jump to Index**. The selected data point is displayed at the top of the reading table.

To save the data, select the menu in the upper left and select **Save to USB**. For information on the options, refer to [Options when saving buffer data to a USB flash drive](#) (on page 6-25).

When TERMINALS is set to **REAR**, you can filter the data using channels and watch channels. The options include:

- **Filter by Watch Channels (Active Buffer):** Filters the data by watch channels. After selecting this option, select **Edit Watch Channels** to select specific channels.
- **Edit Watch Channels (Active Buffer):** Select which channels are watched channels.
- **Filter by Channels:** Allows you to limit the data in the reading table. After selecting **Filter by Channels**, select **Edit Channels** to specify the channels to display.
- **Edit Channels:** Allows you to select the channels that are displayed in the reading table.
- **No Filtering:** Removes filters from the reading table and displays all data for the selected reading buffer.

Using the front panel to view the contents of a reading buffer:

1. Press the **MENU** key.
2. Under **Views**, select **Reading Table**. Data for the active reading buffer is displayed.

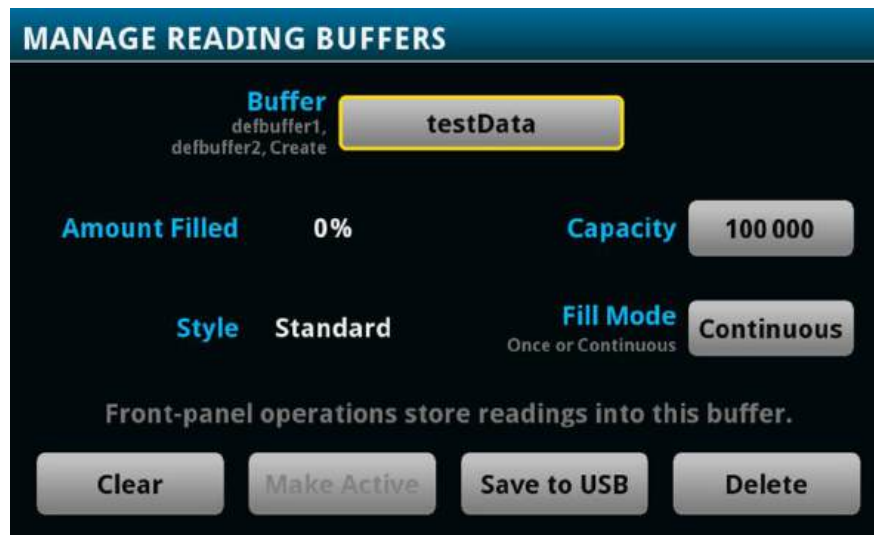
Figure 125: Reading table



3. To display data for a different reading buffer, select the new buffer.
4. To view details for a specific data point, swipe the table up or down and select the data point to view the Reading Details. If there are many data points, select an area on the reading preview graph in the upper right corner of the screen to get closer to the data you want, and then scroll to the data point. You can also select the menu and select **Jump to Index** to go to a specific point.
5. Press the **HOME** key to return to the home screen.

Using the front panel to save buffer content to files:

1. Insert a USB flash drive into the USB port.
2. Press the **MENU** key.
3. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.
4. Select the reading buffer that you want to save. For example, select `testData`.

Figure 126: MANAGE READING BUFFERS window

5. Select **Save To USB**. The File Content dialog is displayed. For information on the options, see [Options when saving buffer data to a USB flash drive](#) (on page 6-25).
6. To change the file name, select **Change**. A keyboard is displayed.
7. Enter the name of the file in which to save the readings.

NOTE

You only need to enter the name of the file you want to save. It is not necessary to enter the file extension. All files are saved as `.CSV` files.

8. Select **OK** on the keyboard.
9. Select **OK** to save the file. When the MANAGE READING BUFFERS window is displayed again, the file is saved.
10. Press the **HOME** key to return to the home screen.

NOTE

You can also save buffer data from the READING TABLE window. Select the menu in the upper left of the READING TABLE window and select **Save to USB**.

Options when saving buffer data to a USB flash drive

The options available when you save buffer data to a flash drive are described in the following table.

Setting	Description
Channel	Include the channel information (number and label) in a column in the saved file.
Extra Value	Available for buffers set to the Full or Writable style. Include extra values, such as the ratio component of a DCV ratio measurement, in the saved file.
File Layout	Determines how data is placed in Microsoft Excel: ■ Reading per Row: Readings are displayed in rows. ■ Reading in Channel Columns: Ignore other columns and use a special format with a column per channel. ■ Spreadsheet Graph: Ignore other columns and use a special format that is easy to graph in Microsoft Excel.
Filename	By default, the file name is the same as the buffer name, followed by the date and time when the file was saved. The date and time are in the format <code>mmdd_hhmmss</code> . To save the file with a different name, select Change . The date and time are not included if you change the file name.
Reading Format	Include the unit, range digits, and display digit settings in the saved file.
Status	Include math, limits, and terminal settings in the saved file.
Time Format	Sets the time format: ■ Absolute: Each timestamp provides the time and date that the reading was made or the number of seconds from the first buffer reading that the reading was made. ■ None: No timestamp. ■ Parts: Timestamps contain dates, hours, minutes, seconds, and fractions of seconds according to Coordinated Universal Time (UTC). ■ Raw: Timestamps display the absolute time in seconds. ■ Relative: Timestamps are oriented to a timer with the first buffer reading timestamped at 0.000000 seconds. Each following timestamp is then based on the presently selected format.

Using the front panel to store readings in the selected buffer

Before you store readings, make sure the correct reading buffer is selected. See [Selecting a buffer](#) (on page 6-17) for more information.

NOTE

Each time a reading buffer is created, the instrument automatically selects the newly created buffer as the active buffer.

To store a reading from the front panel, make a measurement. The buffer-fill indicators light up to indicate that the buffer is filling. Depending on the size of the buffer, the lit indicator may be difficult to observe. When all four indicators are lit, the buffer is completely filled. All the indicators will not be lit if the number of readings stored is less than the selected buffer capacity.

To stop storing readings in a buffer when you are making continuous readings, select the measurement method indicator and select **Manual Trigger Mode**. You can press and hold the **TRIGGER** key for about 3 seconds to display the measurement method options.

NOTE

Stored readings are lost when the instrument is turned off or reset. Stored readings are also lost when you resize a reading buffer.

Using SCPI commands to save or append buffer content to files:

Before using any of these commands, insert a USB flash drive into the USB port.

To save readings and formatted timestamps from the default buffer to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE "/usb1/myData.csv", "defbuffer1"
```

To save readings and formatted timestamps from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE "/usb1/myData.csv", "testData"
```

To append readings and formatted timestamps from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE:APPend "/usb1/myData.csv", "testData"
```

To append readings and formatted timestamps from a reading buffer named `testData` from index 6 to index 10 in file named `myData.csv` on a USB flash drive, send the following command:

```
TRACe:SAVE:APPend "/usb1/myData.csv", "testData", FORM, 6, 10
```

Using TSP commands to save or append buffer content to files:

Before using any of these commands, insert a USB flash drive into the USB port.

To save readings from the default buffer to a file named `myData.csv` on a USB flash drive, send the following command:

```
buffer.save(defbuffer1, "/usb1/myData.csv")
```

To save readings from a reading buffer named `testData` to a file named `myData.csv` on a USB flash drive, send the following command:

```
buffer.save(testData, "/usb1/myData.csv")
```

To append readings from a reading buffer named `testData` with default time information to a file named `myData.csv` on the USB flash drive, send the following command:

```
buffer.saveappend(testData, "/usb1/myData.csv")
```

Clearing buffers

You can clear all readings and statistics from buffers.

If you reset or power down the instrument, data is cleared from all buffers and user-defined reading buffers are removed.

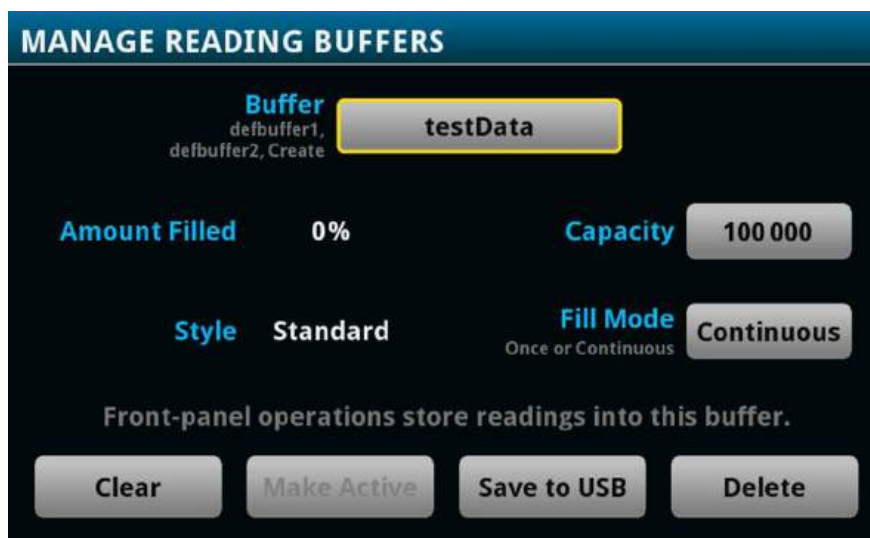
The following topics provide additional information about using the front panel and remote commands to clear buffers.

In addition to the following steps, to clear the active buffer, you can press the **MENU** key and **EXIT** key simultaneously. You can also go to the Statistics swipe screen and select **Clear Active Buffer**.

Using the front-panel Reading Buffers menu to clear a reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.
3. Select a reading buffer from the list. For example, select `testData`.

Figure 127: MANAGE READING BUFFERS window



4. Select **Clear** to clear the buffer.
5. A confirmation message is displayed. Select **Yes**.
6. Press the **HOME** key to return to the home screen.

Using SCPI commands to clear a buffer:

To clear a user-defined buffer named `testData`, send the following command:

```
TRACe:CLear "testData"
```

Using TSP commands to clear a buffer:

To clear a user-defined buffer named `testData`, send the following command:

```
testData.clear()
```

Deleting buffers

If you want to save the readings in a buffer before deleting the buffer, save the buffer to a USB flash drive. See [Viewing and saving buffer content](#) (on page 6-21) for details.

You cannot delete the default buffers `defbuffer1` or `defbuffer2`. However, the data in the default buffers is lost when the instrument is reset or the power is turned off.

Using the front panel to delete a reading buffer:

1. Press the **MENU** key.
2. Under Measure, select **Reading Buffers**. The MANAGE READING BUFFERS window is displayed.
3. Select **Buffer**.
4. Select the buffer to be deleted.
5. Select **Delete** to delete the buffer.
6. When the "Are you sure you want to delete testData" prompt is displayed, select **Yes**.

Using SCPI commands:

To delete a user-defined buffer named `testData`, send the following command:

```
:TRACe:DELeTe "testData"
```

Using TSP commands:

To delete a user-defined buffer named `testData`, send the following command:

```
buffer.delete(testData)
```

NOTE

Do not set the buffer name to `nil` to delete it. To cleanly delete the buffer from the instrument, use the `buffer.delete()` command.

Remote buffer operation

You can control the DAQ6510 buffers through a remote interface using SCPI or TSP remote commands.

This section provides a summary of some of the remote commands available to control and access data stored in buffers; however, this section does not describe all of the available commands. See the following sections for command descriptions:

- For information about SCPI commands, see the [SCPI command reference](#) (on page 12-1)
- For information about TSP commands, see the [TSP command reference](#) (on page 14-1)

Storing data in buffers

Using SCPI commands:

The table below lists the SCPI commands that you use for data storage.

Command	Description
:TRACe:ACTual?	This command contains the number of readings in the specified buffer. See :TRACe:ACTual? (on page 12-229) for more information.
:TRACe:ACTual:END?	This command returns the last index in a reading buffer. See :TRACe:ACTual:END? (on page 12-230) for more information.
:TRACe:ACTual:STArT?	This command returns the starting index in a reading buffer. See :TRACe:ACTual:STArT? (on page 12-231) for more information.
:TRACe:CLear	This command clears all readings and statistics from the specified buffer. See Clearing buffers (on page 6-27) for more information. Also see :TRACe:CLear (on page 12-236).
:TRACe:DELeTe	This command deletes a buffer. See :TRACe:DELeTe (on page 12-240) for more information.
:TRACe:FILL:MODE	This command determines if a reading buffer is filled continuously or is filled once and stops. See :TRACe:FILL:MODE (on page 12-240) for more information.
:TRACe:LOG:STATe	This command indicates whether the reading buffer should log informational events. See :TRACe:LOG:STATe (on page 12-241) for more information.
:TRACe:MAKE	This command creates a user-defined reading buffer. You cannot use this command on the default buffers. See Creating buffers (on page 6-6) for more information. Also see :TRACe:MAKE (on page 12-243).
:TRACe:POINts	This command reads the number of readings a buffer can store. This allows you to change the number of readings the buffer can store. See :TRACe:POINts (on page 12-248) for more information.
:TRACe:SAVE	This command saves data from the specified reading buffer to a USB flash drive. See :TRACe:SAVE (on page 12-249) for more information.
:TRACe:SAVE:APPend	This command appends data from the reading buffer to a file on the USB flash drive. See :TRACe:SAVE:APPend (on page 12-251) for more information.
:TRACe:STATistics:CLear	This command clears the statistical information associated with the specified buffer. This command does not clear the readings.
:TRACe:WRITe:FORMat	For use with writable buffers only; this function sets the units and number of digits that are written into the reading buffer. See :TRACe:WRITe:FORMat (on page 12-264) for more information.
:TRACe:WRITe:READing	For use with writable buffers only; this function writes the data you specify into a reading buffer. See :TRACe:WRITe:READing (on page 12-266) for more information.

Using TSP commands:**CAUTION**

Once you create a reading buffer using TSP commands, if you use that buffer name for another buffer or variable, you can no longer access the original buffer.

The table below lists the TSP commands that you use for data storage.

Command	Description
<code>buffer.clearstats()</code>	This function clears all statistics from the specified buffer. This function does not clear the readings. See buffer.clearstats() (on page 14-16) for more information.
<code>buffer.delete()</code>	This function deletes a user-defined reading buffer. See buffer.delete() (on page 14-17) for more information.
<code>buffer.make()</code>	This function creates a user-defined reading buffer. You cannot use this command on the default buffers. See Creating buffers (on page 6-6) for more information. Also see buffer.make() (on page 14-22).
<code>buffer.save()</code>	This function saves data from the specified reading buffer to a USB flash drive. See buffer.save() (on page 14-27) for more information.
<code>buffer.saveappend()</code>	This function appends data from the reading buffer to a file on the USB flash drive. See buffer.saveappend() (on page 14-29) for more information.
<code>buffer.write.format()</code>	For use with writable buffers only; this function sets the units and number of digits that are written into the reading buffer. See buffer.write.format() (on page 14-34) for more information.
<code>buffer.write.reading()</code>	For use with writable buffers only; this function writes the data you specify into a reading buffer. See buffer.write.reading() (on page 14-37) for more information.
<code>bufferVar.capacity</code>	This attribute reads the number of readings a buffer can store. This allows you to change the number of readings the buffer can store. See bufferVar.capacity (on page 14-40) for more information.
<code>bufferVar.clear()</code>	This function clears all readings and statistics from the specified buffer. See Clearing buffers (on page 6-27) and bufferVar.clear() (on page 14-43) for more information.
<code>bufferVar.fillmode</code>	This attribute determines if a reading buffer is filled continuously or is filled once and stops. See bufferVar.fillmode (on page 14-51) for more information.
<code>bufferVar.logstate</code>	This attribute indicates whether the reading buffer should log informational events. See bufferVar.logstate (on page 14-54) for more information.
<code>bufferVar.n</code>	This attribute contains the number of readings in the specified reading buffer. See bufferVar.n (on page 14-55) for more information.
<code>bufferVar.endindex</code>	This attribute returns the last index in a reading buffer. See bufferVar.endindex (on page 14-45) for more information.
<code>bufferVar.startindex</code>	This attribute returns the starting index in a reading buffer. See bufferVar.startindex (on page 14-60) for more information.

Accessing the data in buffers

Using SCPI commands:

To access a buffer, include the buffer name in the respective command. For example, the following commands:

- Create a buffer named `testData` to store 100 readings
- Set the instrument to make five readings for all measurement requests
- Make the readings and store them in the buffer
- Return five readings (including the measurement and relative time) from the user-defined buffer named `testData`

```
TRAC:MAKE "testData", 100
SENS:COUN 5
TRAC:TRIG "testData"
TRAC:DATA? 1, 5, "testData", READ, REL
```

Using TSP commands:

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

To access a buffer, include the buffer name in the respective command. For example, the following commands:

- Create a buffer named `testData` to store 100 readings
- Set the instrument to make five readings for all measurement requests
- Make the readings and store them in the buffer
- Return five readings (including the measurement and relative time) from the user-defined buffer named `testData`

```
-- Create a buffer named testData to store 100 readings.
testData = buffer.make(100)
-- Set the instrument to make 5 readings and store them in the buffer.
trigger.model.load("SimpleLoop", 5, 0, testData)
-- Make the readings
trigger.model.initiate()
waitcomplete()
-- Read the 5 readings and print them including the measurement
-- and relative time for each reading.
printbuffer(1, 5, testData.readings, testData.relativetimestamps)
```

Buffer read-only attributes

Use buffer read-only attributes to access the information contained in an existing buffer.

Using SCPI commands:

The following commands are available for each reading buffer.

Attribute	Description
:TRACe:ACTual?	This command returns the number of readings in the specified buffer. See :TRACe:ACTual? (on page 12-229) for more information.
:TRACe:ACTual:END?	This command returns the last index in a reading buffer. See :TRACe:ACTual:END? (on page 12-230) for more information.
:TRACe:ACTual:STARt?	This command returns the starting index in a reading buffer. See :TRACe:ACTual:STARt? (on page 12-231) for more information.
:TRACe:DATA?	This command returns the readings stored in a specified reading buffer. See :TRACe:DATA? (on page 12-237) for more information.
:TRACe:STATistics:AVERage?	This command returns average of all readings added to the buffer. See :TRACe:STATistics:AVERage? (on page 12-253) for more information.
:TRACe:STATistics:MAXimum?	This command returns the maximum reading value added to the buffer. See :TRACe:STATistics:MAXimum? (on page 12-255) for more information.
:TRACe:STATistics:MINimum?	This command returns the minimum reading value added to the buffer. See :TRACe:STATistics:MINimum? (on page 12-256) for more information.
:TRACe:STATistics:PK2Pk?	This command returns the peak-to-peak value of all readings added to the buffer. See :TRACe:STATistics:PK2Pk? (on page 12-257) for more information.
:TRACe:STATistics:STDDev?	This command returns the standard deviation of all readings added to the buffer. See :TRACe:STATistics:STDDev? (on page 12-259) for more information.

Using TSP commands:

See [printbuffer\(\)](#) (on page 14-369) for a list of available attributes.

Reading buffer time and date values

Time and date values are represented as a number of UTC seconds since 12:00 a.m. Jan. 1, 1970. Use the following TSP commands to return values in the following formats:

- Hours and minutes: [bufferVar.times](#) (on page 14-62)
- UTC seconds: [bufferVar.seconds](#) (on page 14-59)
- Month-day-year format, or to access the timestamp table: [bufferVar.dates](#) (on page 14-44)

For example, to return the hours and minutes of the readings in `defbuffer1`, send the command:

```
printbuffer(1, 5, defbuffer1.times)
```

The return is similar to:

```
20:30:16, 20:30:16, 20:30:16, 20:30:16, 20:30:16
```

Reading buffer for . . . do loops

The following TSP examples illustrate the use of `for . . . do` loops when recalling data from a reading buffer called `mybuffer`. The following code may be sent as one command line or as part of a script. Example outputs follow the line of code. Also see the [printbuffer\(\)](#) (on page 14-369) command.

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

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

Example comma-delimited output of above code:

```
-1.5794739960384e-09, Amp DC, 0
-1.5190692453926e-11, Amp DC, 0.411046134
-2.9570144943758e-11, Amp DC, 0.819675745
-2.9361919146043e-11, Amp DC, 1.228263492
-3.0666566508408e-11, Amp DC, 1.636753752
-4.0868204653766e-11, Amp DC, 2.034403917
```

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

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

Example columnar-delimited output of above code:

-1.5794739960384e-09	Amp DC	0
-1.5190692453926e-11	Amp DC	0.411046134
-2.9570144943758e-11	Amp DC	0.819675745
-2.9361919146043e-11	Amp DC	1.228263492
-3.0666566508408e-11	Amp DC	1.636753752
-4.0868204653766e-11	Amp DC	2.034403917

Writable reading buffers

Writable reading buffers allow you to add external data manually to a user-defined buffer on the DAQ6510.

You can create a writable buffer by specifying the writable or full writable style when you create the buffer over a remote interface using SCPI or TSP commands. You cannot create a writable buffer from the DAQ6510 front panel.

NOTE

Be aware that when you create a writable buffer, it immediately becomes the active buffer. If you try to save readings from the instrument to the writable buffer, errors occur.

If you switch to front-panel control to make readings after selecting or creating a writable buffer, be sure that you select a buffer that is not of the writable style to be the active buffer before you try to store readings. Writable buffers are for manual entry of user-supplied data only and do not store readings measured by the instrument.

To populate a writable reading buffer, you set the format of the units and the unit values for each buffer index using the following commands:

- SCPI: [:TRACe:WRITe:FORMat](#) (on page 12-264) and [:TRACe:WRITe:READIng](#) (on page 12-266)
- TSP: [buffer.write.format\(\)](#) (on page 14-34) and [buffer.write.reading\(\)](#) (on page 14-37)

After you have populated a writable buffer, you can view the data on your computer from the DAQ6510 virtual front panel or on the front-panel graph screen.

The following example resets the instrument and creates a writable buffer named `writBuffer`. The units for the data are set to Watts and number of display digits to $3\frac{1}{2}$. The example then loads ten lines of data, two with timestamp data and a status marker (256) that shows the data is the first reading in a group, into the buffer.

Using SCPI commands:

```
*RST
TRAC:MAKE "writBuffer", 10, WRIT
TRACe:WRIT:FORM "writBuffer", WATT, 3
TRACe:WRITe:READIng "writBuffer", 1, 0, 0, 256
TRACe:WRITe:READIng "writBuffer", 2
TRACe:WRITe:READIng "writBuffer", 3
TRACe:WRITe:READIng "writBuffer", 4
TRACe:WRITe:READIng "writBuffer", 5
TRACe:WRITe:READIng "writBuffer", 1, 10, 0, 256
TRACe:WRITe:READIng "writBuffer", 2
TRACe:WRITe:READIng "writBuffer", 3
TRACe:WRITe:READIng "writBuffer", 4
TRACe:WRITe:READIng "writBuffer", 5
```


The following example resets the instrument and creates a writable buffer named `writBuffer`. The units for the data are set to Watts and number of display digits to 3½. The example then loads ten lines of data, two with timestamp data and a status marker (`buffer.STAT_START_GROUP`) that shows the data is the first reading in a group, into the buffer.

Using TSP commands:

```
reset()
writBuffer = buffer.make(100, buffer.STYLE_WRITABLE)
buffer.write.format(writBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(writBuffer, 1, 0, 0, buffer.STAT_START_GROUP)
buffer.write.reading(writBuffer, 2)
buffer.write.reading(writBuffer, 3)
buffer.write.reading(writBuffer, 4)
buffer.write.reading(writBuffer, 5)
buffer.write.reading(writBuffer, 1, 10, 0, buffer.STAT_START_GROUP)
buffer.write.reading(writBuffer, 2)
buffer.write.reading(writBuffer, 3)
buffer.write.reading(writBuffer, 4)
buffer.write.reading(writBuffer, 5)
```

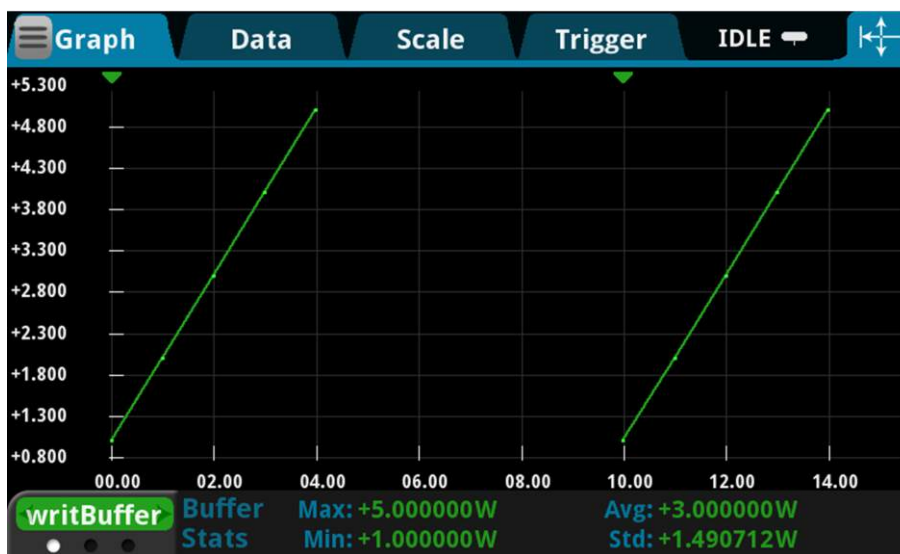
To view the data in the writable buffer on the front-panel graph screen:

1. Press the **MENU** key.
2. Under Views, select **Graph**. By default, time is plotted on the x-axis.
3. Select the **Scale** tab.
4. Set the x-axis method to **Show All Readings**.
5. Select the **Graph** tab.

You can compare the external data to data in another buffer by adding an additional trace to the graph. For more information about graphing data, see [Graphing](#) (on page 7-1).

The graph from the example looks similar to the following figure.

Figure 128: Graph with data from a writable buffer with start group markers



Apply mathematical expressions to reading buffer data

You can apply a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You can apply expressions to readings made from the DMM inputs or readings made from channels. If you have expressions set through both the buffer math and channel math commands, the expressions set for the channel math command take precedence.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style `FULL`. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

Mathematical expressions for buffer math

The expressions you can apply to readings are listed in the following table. In the formulas:

- r = present reading
- a = previous reading
- t = timestamp of the reading
- c = constant

Expression	SCPI parameter TSP parameter	Formula
No math applied	NONE buffer.EXPR_NONE	Not applicable
Add	ADD buffer.EXPR_ADD	$r + a$
Average	AVERage buffer.EXPR_AVERAGE	$\frac{(r+a)}{2}$
Divide	DIVide buffer.EXPR_DIVIDE	$\frac{r}{a}$
Exponent	EXPonent buffer.EXPR_EXPONENT	10^r
Log10	LOG10 buffer.EXPR_LOG10	$\log_{10} r$
Multiply	MULTiply buffer.EXPR_MULTIPLY	$r * a$
Polynomial	POLY buffer.EXPR_POLY	$c0 + c1 \cdot r + c2 \cdot r^2 + c3 \cdot r^3 + c4 \cdot r^4 + c5 \cdot r^5$
Power	POWER buffer.EXPR_POWER	r^c
Rate of change	RATE buffer.EXPR_RATE	$\frac{(r-r_1)}{(t-t_1)}$
Reciprocal	RECiprocal buffer.EXPR_RECIPROCAL	$\frac{1}{r}$
Square root	SQRoot buffer.EXPR_SQROOT	$\sqrt{r}$
Subtract	SUBtract buffer.EXPR_SUBTRACT	$r - a$

Set up buffer math using SCPI commands

The SCPI command for setting buffer math is:

[:TRACe:MATH](#) (on page 12-245)

The SCPI command for setting buffer math for a channel is:

[:TRACe:CHANnel:MATH](#) (on page 12-233)

Set up buffer math using TSP commands

The TSP command for setting buffer math is:

[buffer.math\(\)](#) (on page 14-23)

The TSP command for setting buffer math for a channel is:

[buffer.channelmath\(\)](#) (on page 14-12)

Using buffers across TSP-Link nodes

After connecting two TSP-Link® enabled instruments, you can access buffers over the TSP-Link network.

For local node access to default and custom buffers, you do not need a TSP-Link node number.

For custom buffers on a remote node, you specify the node number when you create the buffer. After the buffer is created, the buffer name is handled as a local variable, so you do not need the node number to refer to the buffer. You can only use that buffer on the remote node on which it was created.

To use the default buffers on a remote node, you need to use the node number in the command to store readings in the default buffer. You also need the node number to access the default buffer data on a remote node.

The following script illustrates how and when you need to include a node reference to access default and custom buffers when the instrument is part of a TSP-Link network.

```
tsplink.initialize()
reset()
-- Access defbuffer1 on the local node.
dmm.measure.read(defbuffer1)
-- Access defbuffer1 on a remote node.
node[9].dmm.measure.read(node[9].defbuffer1)

-- Access a custom buffer on the local node.
myBuffer = buffer.make(100)
dmm.measure.read(myBuffer)
-- Access a custom buffer on a remote node.
myRemoteBufferOnNode9 = node[9].buffer.make(100)
node[9].dmm.measure.read(myRemoteBufferOnNode9)

-- It is illegal to reference the custom buffer on a different node.
-- For example, node[8].dmm.measure.read(myRemoteBufferOnNode9) generates an error.
```

In this section:

Introduction	7-1
About the graph screens	7-2
How to work with the graph	7-5
Use the Graph swipe bar	7-6
Change the data that is graphed	7-9
Add, remove, and clear traces	7-10
Active buffer	7-11
Change the display of data	7-11
Change the scale of the graph	7-12
Set up triggers	7-13
About the Histogram screen	7-18

Introduction

The graphing features of the DAQ6510 allow you to view your measurement data on the front panel of the instrument. The instrument graphs up to twenty traces in an X-Y graph or in a histogram. Each trace represents the data from a reading buffer. You can access each graph individually.

You can manipulate the graph to view minimums and maximums, view averages, determine deltas, and view the values of specific data points. You can also set up triggers and initiate data collection and scans from the graph and histogram screens.

About the graph screens

When you start the instrument, the instrument graphs data from the default reading buffer, defbuffer1. You can change the reading buffer as needed. You can view the graph from either the Graph swipe screen or the Graph screen.

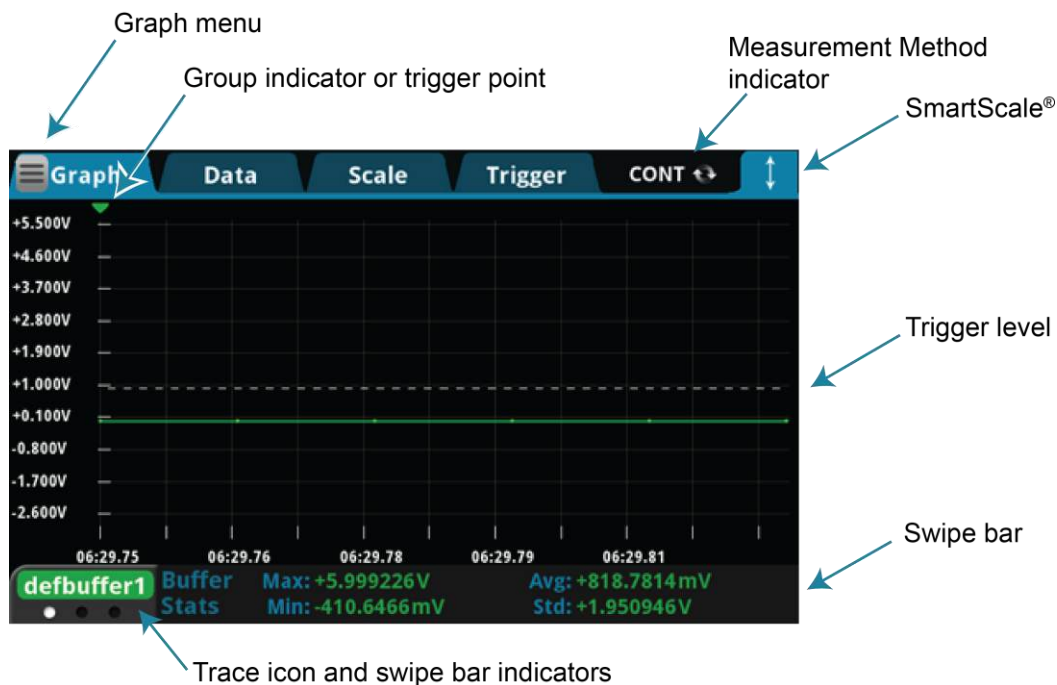
The Graph swipe screen is available from the home screen and displays a smaller version of the full graph. It allows you to see both the data on the home screen and a graph of the information. It does not allow you to zoom in on data or view data at a specific point. To view the graph on the Graph swipe screen, swipe the bottom of the home screen until the graph is displayed. An example of the Graph swipe screen is shown in the following figure.

Figure 129: Home screen with the GRAPH swipe screen displayed



You can open the full graph by selecting the graph icon on the right side of the Graph swipe screen header. You can also open it from the main menu. To open it from the main menu, under Views, select Graph. An example of the full graph is shown in the following figure.

Figure 130: DAQ6510 Graph tab



Use the measurement method indicator in the upper right corner of the screen to select the measurement method. Refer to [Measurement method indicator](#) (on page 3-14) and [Setting up triggers](#) (on page 7-13) for details.

The SmartScale® feature in the upper right allows you to quickly return to automatic scaling. Automatic scaling is turned off if you change the graph by dragging or using pinch to zoom. When the SmartScale feature is on, the instrument keeps the latest data displayed and determines the best way to scale that data based on the data and the instrument configuration (such as the measure count).

The swipe bar at the bottom of the graph displays different types of information about the content of the graph. You can display swipe bars for the scale, buffer statistics, channel statistics, and cursors. The dots below the trace icon show how many swipe bars are available. Refer to [Use the Graph swipe bar](#) (on page 7-6) for detail on the swipe bar options.

If you are using an analog trigger, the trigger level shows the edge or window levels or boundaries. You can adjust the level or boundary on the graph to change the value.

If the count is set to more than one, the group indicator shows the point at which one count ends and another begins. The group indicator does not necessarily mark the location of a trigger event. System latency and programmed delays may cause the first measurement of a group to be displaced in time from its associated trigger event.

The Graph menu in the upper left corner of the screen allows you to manipulate how the data is displayed and tracked on the Graph tab.

The options are:

- **Edit Watch Channels:** Select which channels or the rear terminals to monitor. Each channel is available as a separate trace in the Graph screens. The watch channels are the same throughout the instrument. If you have already set up watch channels on the home screen, they are the same here. Any changes you make to the watch channels here also affect the watch channels used by the histogram and on the home screen.
- **Optimize for Digitizer:** Sets scale values that are optimized for high-speed measurements.
- **Optimize for Measure:** Sets scale values that are optimized for precision measurements.
- **Analog Edge Trigger:** Sets the trigger source event to be an analog trigger with the waveform set to Analog Edge.

How to work with the graph

The Graph tab displays data in an x-y graph. In most cases, the graph shows the timestamp on the x-axis and the measurements on the y-axis. The Graph tab displays the data as it is added to the selected reading buffer.

The timestamps are displayed on the x-axis. When data is coming in quickly, the first timestamp displays the first few digits in orange. Other timestamps show two orange dots (..) in place of those values.

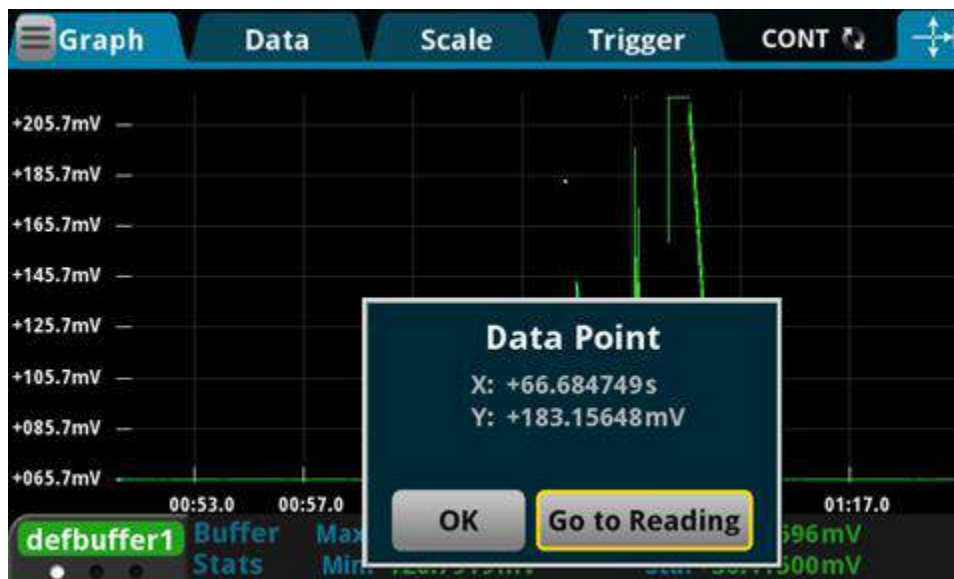
If the y-axis displays a question mark (?), there are multiple units of measure in the reading buffer. Clear the buffer to clear the inconsistent units.

The name of the trace at the bottom left indicates the source of the graph data. If more than one reading is selected as a trace, you can switch between the traces using the trace icon on the lower left. Select the left side of the icon to display the previous trace. Select the right side to display the next trace. Only traces that contain data are displayed.

You can pinch to zoom in the graph. You can also drag the graph to the left or right. When you adjust the view, the SmartScale® feature is turned off. To turn SmartScale on again, select the icon in the upper right of the Graph tab. When SmartScale is on, the instrument determines the best way to scale data based on the data and the instrument configuration (such as the measure count).

You can zoom to display a specific data point. When you select the data point, the Data Point dialog box is displayed with the x and y values of that point. You can also select Go to Reading, which opens the Reading Table screen with that data point selected.

Figure 131: Data point selected on the graph

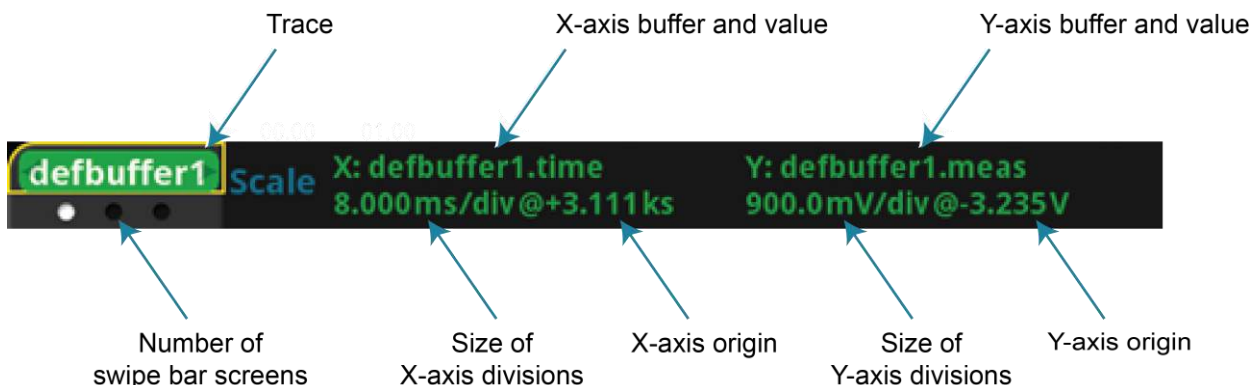


Use the Graph swipe bar

You can swipe the bottom of the Graph tab to display different types of information about the content of the graph. You can display swipe bars for the scale, buffer statistics, channel statistics, and cursors. The dots below the trace icon show how many swipe bars are available.

The Scale swipe bar displays the buffer data that is used for the x and y axes. It also displays the origin value of each axis and the size of the divisions for each axis in the present view.

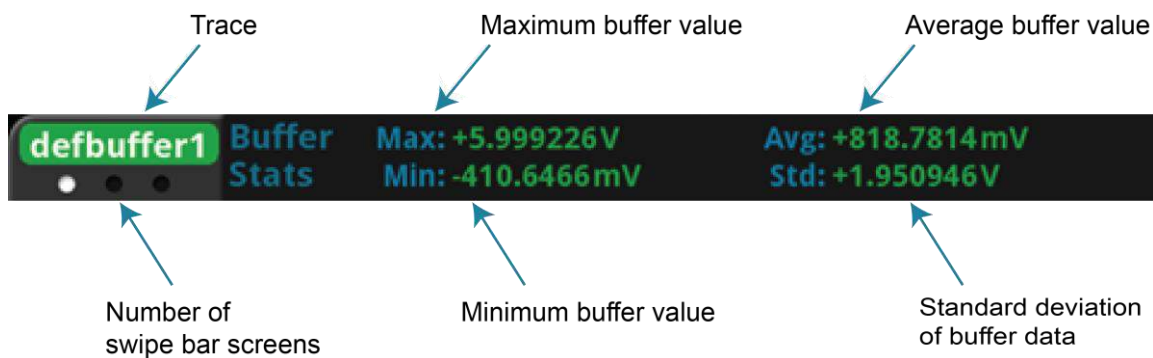
Figure 132: Scale swipe bar



The Buffer Stats swipe bar displays statistics for the readings in the selected trace. If vertical cursors are displayed, the statistics reflect the value within the cursors. You can move the cursors to change the reported statistics. The statistics are:

- **Max:** Maximum value.
- **Min:** Minimum value.
- **Avg:** Average of the values.
- **Std:** Standard deviation for the buffer.
- **Pk2Pk:** Shown instead of standard deviation if vertical cursors are selected. The deviation between the peak-to-peak values.

Figure 133: Buffer Stats swipe bar



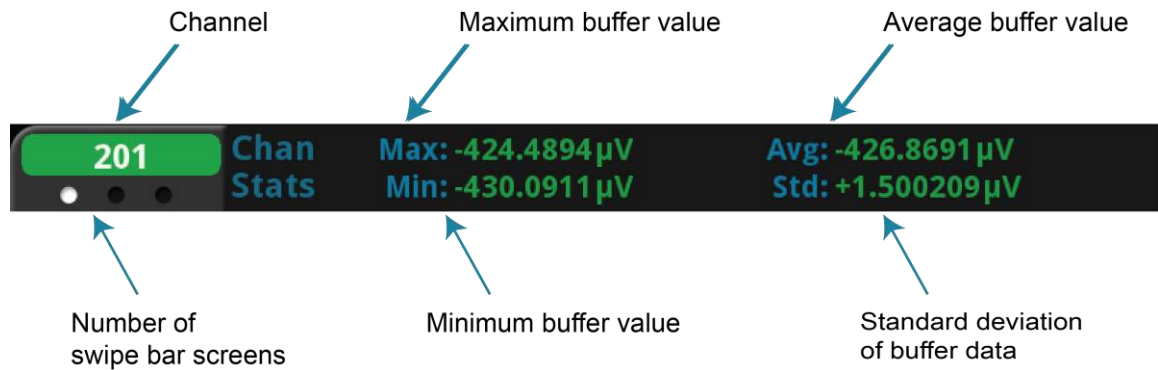
If the buffer type is set to full and contains extra values, such as when the function is set to DCV Ratio, the statistics are shown as Not Applicable.

If the trace is displaying data for a single channel, the Chan Stats swipe bar is displayed instead of the Buffer Stats swipe bar. The statistics are for the values tracked for the channel.

NOTE

If cursors are selected, the Chan Stats swipe bar is replaced by the VCursor Stats swipe bar. To display the Chan Stats swipe bar, turn cursors off.

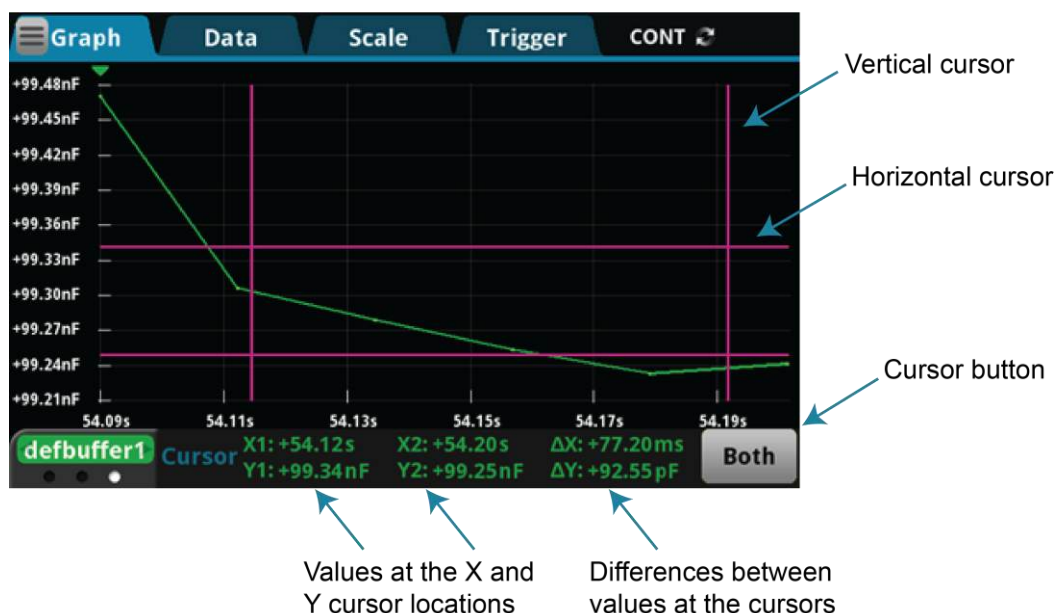
Figure 134: Chan Stats swipe bar



The Cursor swipe bar allows you to display cursors on the graph. If the cursors are displayed, the swipe bar also displays the value at each cursor and the differences between the values at the cursors. You can display vertical, horizontal, or both cursors.

When cursors are displayed, you can drag them to change their positions. You can also move the graph behind the cursors. To move the graph, select a portion of the graph that is not near a cursor and drag.

Figure 135: Cursor swipe bar



When cursors are enabled, they are displayed for all traces.

Change the data that is graphed

You can change the data that is displayed on the Data tab of the Graph view. On the Data tab, you can:

- Add, remove, and clear traces
- Change how data is displayed

When you change the traces assigned to the histogram, it also changes the traces assigned to the graph. Conversely, changing the traces for the graph changes the traces assigned to the histogram.

Add, remove, and clear traces

The graph plots data from reading buffers, which are set up on the Data tab as traces. The data from each buffer is shown as a separate trace on the Graph tab. You can select up to 20 traces. The selected traces are shown in the Traces list on the Data tab. If the reading buffer contains data from channels, you can select channels as traces. Each channel is plotted on one trace.

To add a trace, select Add. If the reading buffer does not contain additional data, the buffer is added to the Traces list.

If the reading buffer contains extra data, you are prompted to select which buffer element to graph. Select Measure to plot the measurement values or Extra to plot the extra values. Extra values are available when the buffer type is set to Full or Full Writable.

If the reading buffer contains channel data, you are prompted to select the channel that contains the values you want to plot. Only one channel can be selected for each trace. If the channel is assigned a label, the label is displayed after the buffer name in the Traces list and in the Traces icon on the graph.

If you have 20 traces in the list, Add is not available. You must remove a trace before you can add another one.

Colors are automatically assigned to the traces and cannot be changed.

To remove a trace, select the trace and select Remove. The trace is removed from the graph. The data in the buffer is not affected.

To remove data from a reading buffer, select the reading buffer from the Traces list and select Clear Buffer. To clear data from the active buffer, you can press the MENU and EXIT keys simultaneously.

Active buffer

The active buffer contains the data that is displayed on the home screen and where readings are stored when Continuous Measurement is selected or a manual trigger is generated.

When an active buffer is selected on the Data tab, that trace tracks the active buffer instead of a specific buffer. If the active buffer changes, the data that is displayed changes to match the new active buffer.

To remove the active buffer from the list of traces, select it and select Remove Trace. This does not affect the active buffer that is selected on the home screen. To restore the active buffer to the list of traces, select Add and select the trace labeled Active.

When the Terminals switch is set to Rear, Watch Channels is displayed instead of Active.

Change the display of data

You can set the drawing style and the graph type of the graph on the Data tab of the Graph screens.

The drawing style determines how data is represented when there are many data points. You can select Line, Marker, or Both.

When Line is selected, the data points are connected with solid lines. When Marker is selected, the individual data points are shown with no connecting lines. When Both is selected, the individual data points are shown, and the points are connected with solid lines.

The Graph Type sets the data to be plotted on the x-axis for all traces. You can select Scatter or Time. Scatter is only available if the buffer style is set to Full. All traces must be based on buffers that are set to full in order to select Scatter.

Change the scale of the graph

The Scale tab contains settings that allow you to fine-tune the display of data on the Graph tab. You can select automatic scaling or specific values for the x and y axes.

For both axes, you can select the SmartScale® feature. When the SmartScale feature is selected, the instrument scales the graph automatically, determining the best scaling and tracking method based on the data, reading groups, number of traces, and instrument configuration. The scale is set to show the most relevant portion of the data that is in the selected reading buffer.

For the x-axis, you can also set the Method to one of the following options:

- **Show New Readings:** The graph always displays the latest data on a fixed scale.
- **Show Group of Readings:** The graph displays a group of readings. A group is automatically created when the measure or digitize count is set to more than 1.
- **Show All Readings:** All data in the buffer is displayed on the graph.

If you are graphing one trace, you can set the y-axis to one of the following methods:

- **Autoscale Always:** Continuously scales the y-axis of the trace so it fits the height of the screen.
- **Autoscale Once:** Scales the y-axis of the trace once.

If you are graphing more than one trace, you can set the Y-Axis Method to one of the following options:

- **Independent Autoscale:** Scales the y-axis of the trace so it fits the height of the screen.
- **Swim Lanes:** Scales the y-axis of the traces in equal, nonoverlapping portions of the height of the screen.
- **Shared Autoscale:** Scales the y-axis so that the minimum and the maximum are shared across all traces. Accommodates the minimum and maximum of all traces.

You can also set the automatic scaling method to off for either axis. When automatic scaling is off, you can manually set the scale and the minimum position. For the y-axis, you can set the scale for each trace. Use the Trace button above the Scale and Minimum Position options to select the trace. Information specific to the selected trace is shown in the same color as the trace.

Scale sets the reading value scale for each division. To select a scale, chose Up or Down until the scale you want to set is displayed, then select the value. The scale changes to show the new value.

Minimum Position sets the first value that is visible on the graph for the selected trace.

The Y-Axis Scale Format determines if the graph is linear or logarithmic. Select Linear to increase the step size in even increments. Select Log to increase the step size exponentially.

Set up triggers

The Source Event option on the Trigger tab lets you define the trigger events and attributes that initiate system measurements.

When you set up triggers through the Trigger tab, the instrument defines the LoopUntilEvent trigger-model template with the trigger settings. The active reading buffer is used as the buffer for the trigger model. If a trigger model exists, it is replaced by the new settings.

To use the configured source event for triggered measurements, you need to change the measurement method from Continuous to Initiate Trigger Model. You can do this while on the Trigger tab. Select the measurement method indicator (to the right of the Trigger tab) and select Initiate Trigger Model.

When you set up the source event, settings are applied as you select the options for the selected source event. The settings are retained and displayed when you return to the Trigger tab.

Types of triggers

You can set triggers to be generated from the:

- **Digital Input Line:** The trigger occurs when a pulse is detected from the digital input line.
- **TSP-Link Input:** The trigger occurs on a falling, rising, or either edge of the TSP-Link input.
- **Display TRIGGER Key:** The trigger occurs when you press the TRIGGER key.
- **External Trigger In:** The trigger occurs when an external pulse is detected. The external pulse can come from a digital input line, TSP-Link input line, or the rear-panel external input line.
- **Waveform:** Select an analog edge or window to trigger. Analog triggers are only available for the DC voltage, DC current, digitize voltage, and digitize current functions.

Select None to clear the setup for the source event trigger.

NOTE

The digital and TSP-Link options require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

NOTE

This section describes in general how to set up triggering. It does not describe details on the trigger sources.

- For detail on the digital I/O, refer to [Digital I/O](#) (on page 8-1).
- For detail on external inputs and output, refer to [External trigger control](#) (on page 8-16).
- For detail on TSP-Link, refer to [TSP-Link System Expansion Interface](#) (on page 9-1).
- For detail on waveforms, refer to [Analog triggering overview](#) (on page 8-20).

Trigger settings

For all triggers, you can set the delay, position, and trigger clear options. Some of the triggers have additional settings. All the settings are described in the following text.

Source Event

The Source Event selects the type of trigger.

When you select the source event, you may be prompted for additional information:

- When you select the Digital Input Line, you are prompted for the input line that generates the trigger (1 to 6).
- When you select TSP-Link Input, you are prompted for the TSP-Link input line that generates the trigger (1 to 3).
- When you select Waveform, you are prompted to select Analog Edge or Analog Window. Refer to [Waveform](#) (on page 7-16) for more information on the analog options.

To change the line or waveform, select the source event again.

Settings

The Settings icon is available when the source is set to External Trigger In, Digital Input Line, or TSP-Link Input. Select the icon to set the type of edge that generates a trigger. You can set rising, falling, or either.

Delay

The Delay is the length of time that occurs before the measurement occurs after detecting the selected source event trigger. You can select from 167 ns to 10 ks. Select 0 to set no delay.

Position

The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled.

Trigger Clear

Trigger Clear determines if triggers are cleared before the wait period for the trigger begins. The wait period is set in the trigger model as the wait block.

If you set Trigger Clear to Never, the trigger model clears triggers for the wait block when the trigger model is initiated. The instrument begins making measurements as soon as the trigger model reaches the wait block if it detected the event after being initiated and before reaching the wait block. If the trigger was not detected, the trigger model waits to detect the event before making measurements.

If you set Trigger Clear to Enter, any triggers that occurred after the trigger model was initiated and before reaching wait block are cleared. The source event trigger must occur after reaching the wait block before measurements will begin.

All triggers are cleared when the trigger model begins. Trigger Clear only affects triggers that occur after the trigger model is initiated. Triggers are also cleared when the trigger model exits the Wait block.

Waveform

When the analog waveform is set as the source event, there are several additional settings that are available. Analog waveforms are available for the DC current, DC voltage, digitize current, or digitize voltage functions.

NOTE

If the function is DC current or DC voltage, before setting up the waveform, set the measure range to a value (it cannot be set to Auto) and set Auto Zero to Off. Select the **MENU** key, then select **Settings** to change the settings.

You can select one of the following waveforms:

- Edge: The trigger event occurs when the signal crosses a certain level.
- Window: The trigger event occurs when the signal enters or exits a window that is defined by low and high signal levels.

NOTE

If you have a fast cyclic signal, the trigger may occur before the instrument can gather enough pretrigger data. If this occurs, you see less pretrigger data than expected. However, the correct amount of post-trigger data is collected.

For additional detail on waveforms, refer to [Analog triggering overview](#) (on page 8-20).

Edge settings

When you set Waveform to Edge, you can set the level and the slope in addition to the delay, position, and trigger clear settings.

Level is the signal level that generates the trigger event. The level can be set to any value within the selected measurement range.

The Slope defines if the instrument watches for a rising or falling edge. When Rising is selected, the trigger event is generated when the analog signal trends from below the analog signal level to above the level. When Falling is selected, the trigger event is generated when the signal trends from above to below the level.

Window settings

When you set Waveform to Window, you can set boundaries for the window and the direction in addition to the delay, position, and trigger clear settings.

The window is defined by two levels. The Low Boundary is the lower boundary of the analog trigger window. The High Boundary is the upper boundary of the analog trigger window. The high boundary must be higher than the low boundary.

The Direction determines if the trigger occurs when the signal enters or exits the window. Select Entering if the analog trigger occurs when the signal enters the window. Select Leaving if the analog trigger occurs when the signal exits the window.

Graph measurement using triggers

To set up triggers to occur when a trigger occurs:

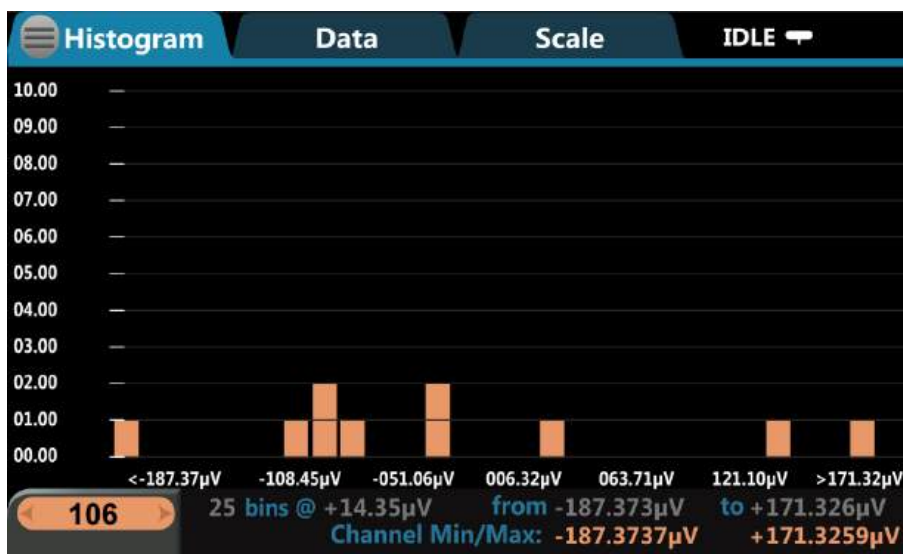
1. Press the **MENU** key.
2. In the View menu, select **Graph**.
3. Select the **Trigger** tab.
4. Set the **Source Event**.
5. Make other settings as needed.
6. Select the **measurement method** indicator at the upper right of the screen and select **Initiate Trigger Model** or **Initiate Scan**.
7. To start the measurements, generate the trigger event.
8. Select the **Graph** tab to view the readings.

About the Histogram screen

The Histogram tab graphs readings as a bar graph of the data distribution into bins. To view the histogram, select the **MENU** key, then select **Histogram**.

An example of the histogram screen is shown in the following figure.

Figure 136: Histogram screen



When you start the instrument, the instrument bins data from the default reading buffer defbuffer1. You can change the reading buffer as needed.

The measurement method indicator in the upper right of the screen selects the measurement method. Refer to [Measurement method indicator](#) (on page 3-14) and [Set up triggers](#) (on page 7-13) for details.

The Histogram menu in the upper left of the screen allows you to select the Edit Watch Channels option. You can select which channels to monitor. Each channel is available as a separate trace on the Histogram.

The watch channels are the same throughout the instrument. If you have already set up watch channels on the home screen, they are the same here. Any changes you make to the watch channels here also affect the watch channels used by the graph and on the home screen.

How to work with the Histogram

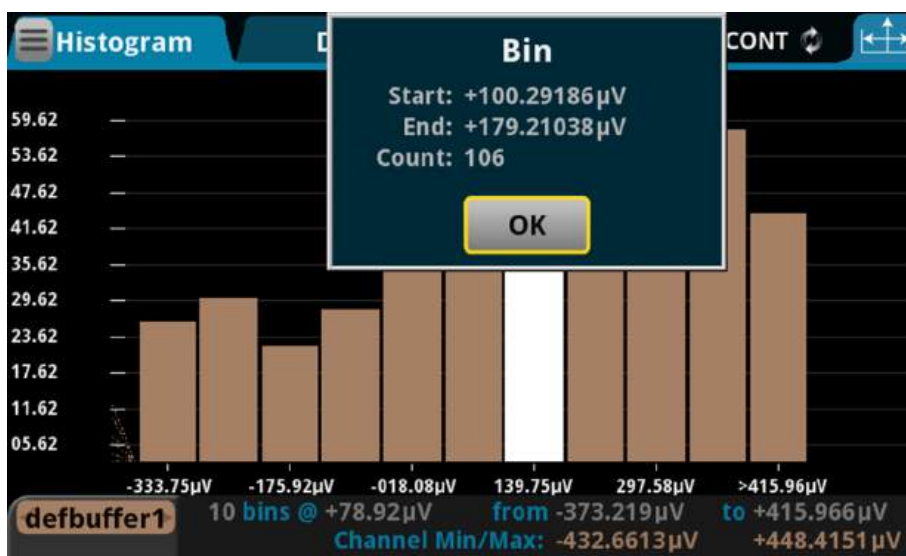
The Histogram tab groups data into bins as it is added to the reading buffer for the selected trace. The range for each bin is shown on the x-axis. The number of readings is shown on the y-axis.

The trace icon at the bottom left indicates the source of the graph data. If more than one reading buffer is selected as a trace, you can switch between the traces using the trace icon. Select the left side of the icon to display the previous trace. Select the right side to display the next trace. Only traces that contain data are displayed.

You can pinch to zoom into or out of data on the histogram. You can also swipe the histogram to the left or right. When you adjust the view, the SmartScale® feature is turned off. To turn SmartScale on again, select the icon in the upper right of the Graph tab. When SmartScale is on, the instrument determines the best way to scale data based on the data and the instrument configuration.

You can select a bin to display the detail for that bin. When you select a bin, the Bin dialog box is displayed with the Start and End values of the bin and the number of measurements within that range.

Figure 137: Histogram Bin dialog box



If the top of a bin has a brighter color rectangle, there is additional data in the bin that is off the screen.

The histogram legend displays information about that data, including the number of bins and the total data range. It also displays the minimum and maximum values of the data in the buffer.

Change the data that is binned

You can change the data that is displayed on the Data tab of the Graph view. On the Data tab, you can add, remove, and clear traces.

NOTE

When you change the traces assigned to the graph, it also changes the traces assigned to the histogram. Conversely, changing the traces for the histogram changes the traces assigned to the graph.

Add, remove, and clear traces

The histogram plots data from reading buffers, which are set up on the Data tab. The data from each buffer is shown as a separate trace on the Histogram tab. You can select up to 20 traces. The selected traces are shown in the Traces list in the Data tab.

To add a trace, select Add and select the buffer. If the buffer contains channel data, you are prompted to select the channel that contains the values you want to plot. The buffer or channel is added to the Traces list. If the buffer contains channels have been assigned labels, the labels are displayed after the buffer name in the Traces list and in the Traces icon on the graph.

If you have 20 traces in the list, Add is not available. You must remove a trace before you can add another one.

Colors are automatically assigned to the traces and cannot be changed.

To remove a trace, select the trace and select Remove. The trace is removed from the graph. The data in the buffer is not affected.

To remove data from a reading buffer, select the reading buffer from the Traces list and select Clear Buffer. To clear the active buffer, you can press the MENU and EXIT keys simultaneously.

Active buffer

The active buffer contains the data that is displayed on the home screen and where readings are stored when Continuous Measurement is selected or a manual trigger is generated.

When an active buffer is selected on the Data tab, that trace tracks the active buffer instead of a specific buffer. If the active buffer changes, the data that is displayed changes to match the new active buffer.

To remove the active buffer from the list of traces, select it and select Remove Trace. This does not affect the active buffer that is selected on the home screen. To restore the active buffer to the list of traces, select Add and select the trace labeled Active.

When the Terminals switch is set to Rear, Watch Channels is displayed instead of Active.

Change the scale of the histogram

The Scale tab contains settings that allow you to fine-tune the display of data on the Histogram tab. You can select automatic scaling or specific values for the bins and boundaries.

The SmartScale® feature scales the histogram automatically. The instrument determines the best way to bin the data.

The Auto Bin option redistributes the data evenly in the bins based on the present minimum and maximum boundaries.

The Fit option adjusts the y-axis scale so that the tops of all bins are visible.

You can also set the automatic scaling method off. When automatic scaling is off, you manually set the minimum and maximum boundaries. The Maximum Boundary is the highest value of the data that is binned in the histogram. Data that is above this level is binned in the high outlier bin. The Minimum Boundary is the lowest value of the data that is binned in the histogram. Data that is below this level is binned in the low outlier bin.

The Number of Bins determines how many bins the data are sorted into. The histogram creates two outlier bins in addition to the bins you define. These bins are used to collect data that is below or above the defined minimum and maximum boundaries.

Triggering

In this section:

Digital I/O	8-1
External trigger control	8-16
Triggering	8-18
Trigger model	8-35

Digital I/O

NOTE

This section applies only if you have installed the Keithley Instruments KTTI-GPIB, KTTI-TSP, or KTTI-RS232 communications accessory.

The DAQ6510 digital I/O port provides six independently configurable digital input/output lines.

You can use these lines for digital control by writing a bit pattern to the digital I/O lines. Digital control is used for applications such as providing binning codes to a component handler. Digital control uses the state of the line to determine the action to take.

You can also use these lines for triggering by using the transition of the line state to initiate an action. The instrument can generate output trigger pulses and detect input trigger pulses. Triggering is used for applications such as synchronizing the operations of a measurement instrument with the operations of other instruments.

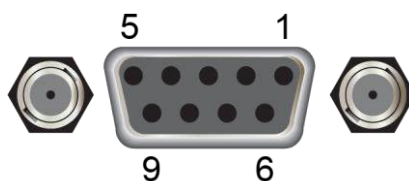
You cannot configure or directly control the digital I/O lines from the front panel. To configure and control any of the six digital input/output lines, you need to send commands to the DAQ6510 over a remote interface. You can use either the SCPI or TSP command set.

See [Remote communications interfaces](#) (on page 2-7) for information about setting up a remote interface and choosing a command set.

Digital I/O connector and pinouts

The digital I/O port uses a standard female DB-9 connector. You can connect to the DAQ6510 digital I/O using a standard male DB-9 connector. The port provides a connection point to each of the six digital I/O lines and other connections, as shown in the following table.

Figure 138: Digital I/O communications port



DAQ6510 digital I/O port pinouts

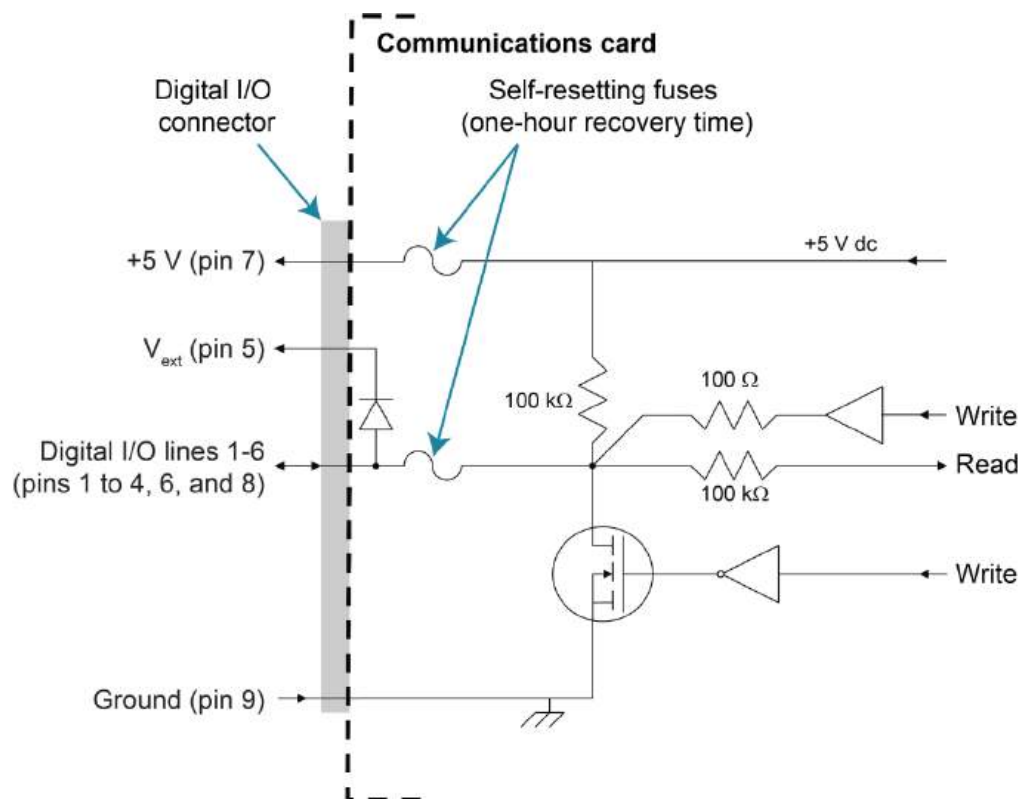
Pin	Description
1	I/O line #1
2	I/O line #2
3	I/O line #3
4	I/O line #4
5	V _{ext} line (relay flyback diode protection; maximum 33 V)
6	I/O line #5
7	+5 V line. Use this pin to drive external logic circuitry. Maximum current output is 500 mA. This line is protected by a self-resetting fuse (one-hour recovery time).
8	I/O line #6
9	Ground

Digital I/O port configuration

The following figure shows the basic configuration of the digital I/O port.

To set a line high (nominally +5 V), write a 1 to it; to set a line low (nominally 0 V), write a 0 to it. To allow an external device to control the state of the line, the line must be set to input mode or open-drain mode. An attached device must be able to sink at least 50 μA from each I/O line.

Figure 139: Digital I/O port configuration



Vext line

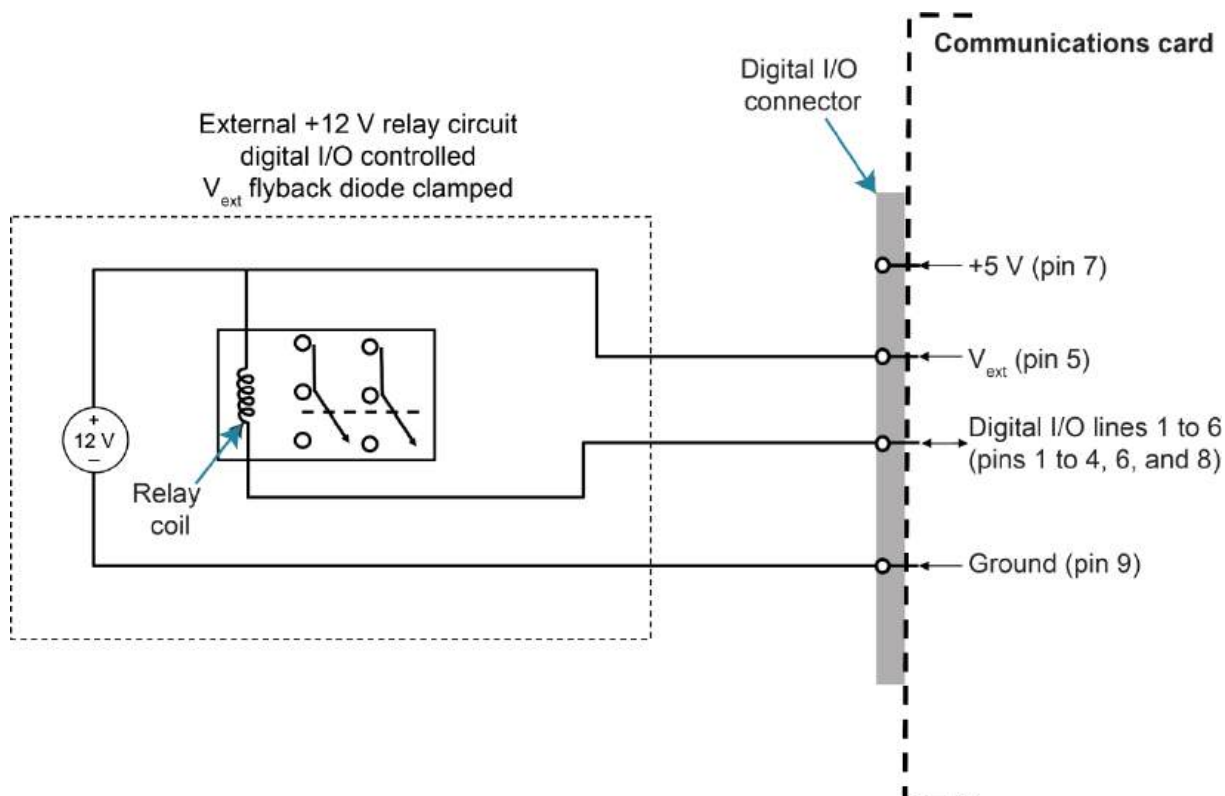
The digital I/O port provides a line (V_{ext}) with a flyback diode clamp that you can use when controlling inductive circuitry such as relay coils or low-power solenoids. You can use the built-in 5 V supply or an external voltage supply for these types of applications. The externally supplied voltage can be up to +33 V.

CAUTION

Do not apply more than 50 mA (maximum current) or exceed +33 V (maximum voltage) on the digital I/O lines. Applying current or voltage exceeding these limits may damage the instrument.

Refer to the following figure for a simplified schematic of a sample control circuit for a relay. You can externally power a different device by replacing the relay coil with the other device. The relay is actuated by configuring the corresponding digital output line. Most of these types of applications use an active-low (set the bit to 0) to turn the relay on (ON = 0 V). In the low state (0 V), the output transistor sinks current through the external device. In the high state, the output transistor is off (transistor switch is open). This interrupts current flow through the external device.

Figure 140: Digital I/O port (example external circuit)



+5 V line

The digital I/O port provides a +5 V output. You can use this line to drive external circuitry. The maximum current output for this line is 500 mA. A self-resetting fuse with a one-hour recovery time protects this line.

If you are using this supply to drive a relay, it should be connected to the V_{ext} line so that the relay is protected by the flyback diode clamp.

Digital I/O lines

You can place each digital I/O line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or acceptor

NOTE

When you configure the digital I/O lines for triggering applications, configure the output lines before the input lines. This prevents possible false input trigger detection in certain situations.

Digital control modes

If you are setting a line for digital control, you can set the line to be open-drain, output, or input, as described in the following topics.

Open-drain

When you place a line in open-drain mode, the line is configured to be an open-drain signal with a 100 k Ω pull-up resistor. This makes the line compatible with other instruments that use open-drain digital I/O lines, such as other Keithley products that only support open-drain for its digital I/O. In this mode, the line can serve as an input, an output, or both. You can read from the line or write to it. When a digital I/O line is used as an input in open-drain mode, you must write a 1 to the line to enable it to detect logic levels that are generated from external sources.

Output

When you place a line in output mode, you can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low. Unlike the input or open-drain modes, it will not respond to externally generated logic levels.

When you read the line, it shows the present output status and an event message is generated.

Input

The input mode is similar to the open-drain mode, except that a line in this mode is intended to be used strictly as an input. When you place a line in input mode, the instrument automatically writes a 1 to the line to enable it to detect externally generated logic levels.

You can read an input line, but you cannot write to it. You also cannot change the logic level while the line is in input mode. If you attempt to change the logic level of a line that is in input mode, an event message is generated.

Trigger control modes

You can use the trigger control modes to synchronize instrument operation with the operation of other instruments. These modes either detect or generate transitions in the state of the line, from high to low (falling edge) or from low to high (rising edge). The input edge detection setting of the instrument determines which type of transition is detected as an input trigger. Output triggers are typically generated in the form of a pulse. The type of transition that occurs on the leading edge of the pulse is determined by an output logic setting. The duration of the pulse is determined by a pulse width setting.

You can use the trigger control modes with interactive triggering or with the trigger model. For more information about the trigger modes and triggering, refer to [Triggering](#) (on page 8-18).

Open-drain

When you set the instrument to trigger mode and place a line in open-drain mode, the line is configured to be an open-drain signal with a 100 k Ω pull-up resistor. This makes the line compatible with other instruments that use open-drain trigger signals, such as other Keithley products that only support open-drain for its digital I/O. In this mode, you can use the line to detect input triggers or generate output triggers, or both. To use this mode successfully, you must carefully configure the input edge and output logic settings because both of these affect the initial state of the trigger line. It is recommended that you reset the line before selecting and configuring this mode.

To use the line only as a trigger input:

1. Reset the line.
2. Set the input trigger edge detection type to falling, rising, or either.

The command that sets the detection type automatically sets the line high. This enables the line to respond to and detect externally generated triggers.

Do not set the output trigger logic type to positive after setting the edge detection type. This sets the line low, which will prevent the line from operating correctly as a trigger input.

To use the line only as a trigger output:

1. Reset the line.
2. Set the output trigger logic type to negative (falling edge) or positive (rising edge).

When you set the logic type to negative, the instrument automatically sets the line high. Setting the logic type to positive automatically sets the line low.

Do not set the input trigger edge detection type after setting the positive logic type. This will set the line high, which will prevent the line from operating correctly as a trigger output.

To use the line as both a trigger input and a trigger output (falling edge triggers only):

1. Reset the line.
2. Set the output trigger logic type to negative (falling edge).
3. Set the input trigger edge detection type to falling, rising, or either.

You can use these settings for triggering applications that use Keithley products offering Trigger Link.

Output

When you place a line in output mode, it is automatically set high or low depending on the output logic setting. Use the negative logic setting when you want to generate a falling edge trigger. Use the positive logic setting when you want to generate a rising edge trigger. You cannot detect incoming triggers on a line configured as a trigger output.

Input

When you place a line in input mode, it is automatically set high to allow it to respond to and detect externally generated triggers. Depending on the input edge detection setting, the line can detect falling-edge triggers, rising-edge triggers, or both.

The line cannot generate an output trigger if it is set to the trigger input mode.

Synchronous triggering

The synchronous triggering modes allow you to:

- Implement bidirectional triggering on a single trigger line
- Start operations on one or more external instruments using a single trigger line
- Wait for all instruments to complete all triggered actions

To coordinate non-Keithley instrumentation with synchronous triggering, the non-Keithley instrument must have a trigger mode that is similar to the synchronous acceptor or synchronous master trigger mode.

To use synchronous triggering, configure the triggering master to synchronous master trigger mode or the non-Keithley equivalent. Configure all other instruments in the test system to the synchronous acceptor trigger mode or equivalent.

Synchronous master

Use the synchronous master trigger mode with the synchronous acceptor mode or its non-Keithley equivalent.

Configure only one instrument as a synchronous master. Configure all other instruments that are connected to the synchronization line as synchronous acceptors.

When a digital I/O line is set to the synchronous master mode, it generates falling edge output triggers and detects rising edge input triggers on the same trigger line.

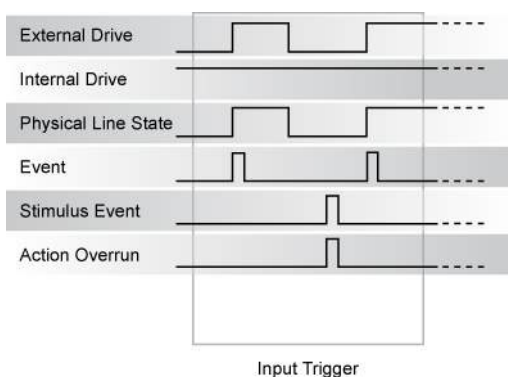
Instruments that are configured as synchronous acceptors detect the falling-edge trigger and begin their triggered actions. At the same time, they latch the line low and hold it in that state until their triggered actions complete. Each instrument configured as an acceptor releases the line upon completion of its triggered actions.

When all instruments have released the line, the line changes state and generates a rising edge trigger. This trigger is detected by the synchronous master, which then performs its next triggered action.

Input characteristics:

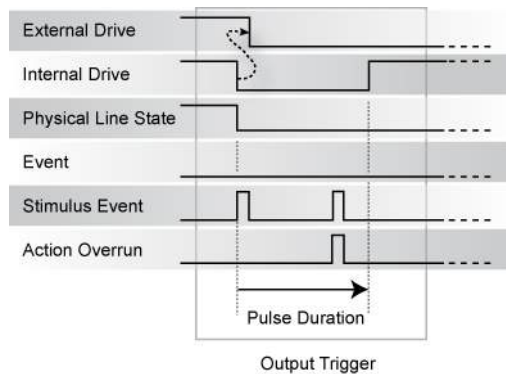
- All rising edges are input triggers.
- When all external drives release the physical line, the rising edge is detected as an input trigger.
- A rising edge is not detected until all external drives release the line and the line floats high.

Figure 141: Synchronous master input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the TSP commands `trigger.digout[N].assert()` and `trigger.tsplinkout[N].assert()` generate a low pulse that is similar to the falling-edge trigger mode.
- An action overrun occurs if the physical line state is low when a stimulus event occurs.

Figure 142: Synchronous master output trigger**Synchronous acceptor**

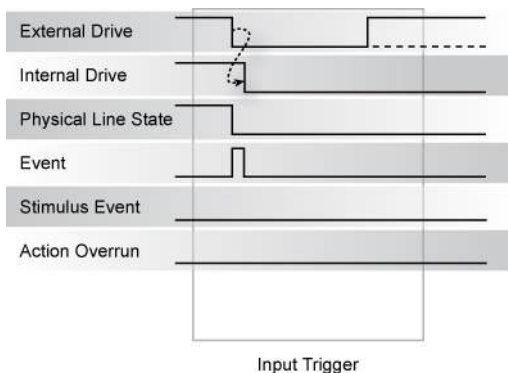
Use the synchronous acceptor trigger mode with the synchronous master mode or its non-Keithley equivalent.

Only one instrument should be configured as a synchronous master. All other instruments connected to the synchronization line must be configured as synchronous acceptor or equivalent.

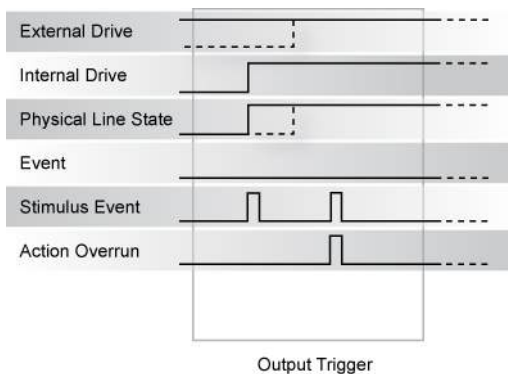
A line that is set to the synchronous acceptor mode detects falling edge input triggers and generates rising edge output triggers on the same trigger line. When a line that is configured as synchronous acceptor detects the falling edge trigger, it latches the line low and holds it in that state until all triggered actions for that instrument are complete. When the triggered actions are complete, the synchronous acceptor line releases the line. When all connected instruments have released the line, the line changes state and generates a rising edge 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 143: Synchronous acceptor input trigger**Output characteristics:**

- In addition to trigger events from other trigger objects, the TSP commands `trigger.digout[N].assert()` and `trigger.tsplinkout[N].assert()` also trigger events.
- The physical line state does not change until all drives (internal and external) release the line.
- Action overruns occur if the internal drive is not latched low and a source event is received.

Figure 144: Synchronous acceptor output trigger

Remote digital I/O commands

Commands for both SCPI and TSP are summarized in the following table. You can use the digital I/O port to do the following actions:

- Perform basic steady-state digital I/O operations, such as reading and writing to individual I/O lines or reading and writing to the entire port
- Trigger the instrument when external trigger pulses are applied to the digital I/O port
- Provide trigger pulses to external devices

SCPI command TSP command	Description
:DIGital:LINE<n>:MODE (on page 12-44) digio.line[N].mode (on page 14-112)	This command sets the mode of the digital I/O line to be a digital line, trigger line, or synchronous line and sets the line to be input, output, or open-drain.
A line reset is not available in SCPI; however, the line is reset when a global reset (*RST) is sent digio.line[N].reset() (on page 14-114)	This command resets digital I/O line values to their factory defaults.
:DIGital:LINE<n>:STATe (on page 12-46) digio.line[N].state (on page 14-116)	This command sets a digital I/O line high or low when the line is set for digital control and returns the state on the digital I/O lines.
:DIGital:READ? (on page 12-47) digio.readport() (on page 14-116)	This command reads the digital I/O port. All six lines must be configured as digital control lines. If not, this command generates an error.
:DIGital:WRITe <n> (on page 12-48) digio.writeport() (on page 14-117)	This command writes to all digital I/O lines. All six lines must be configured as digital control lines. If not, this command generates an error.
:TRIGger:DIGital<n>:IN:CLEar (on page 12-303) trigger.digin[N].clear() (on page 14-448)	This command clears the trigger event on a digital input line.
:TRIGger:DIGital<n>:IN:EDGE (on page 12-303) trigger.digin[N].edge (on page 14-449)	This command sets the edge used by the trigger event detector on the given trigger line.
:TRIGger:DIGital<n>:IN:OVERrun? (on page 12-304) trigger.digin[N].overrun (on page 14-450)	This command returns the event detector overrun status.
Not available in SCPI trigger.digin[N].wait() (on page 14-451)	This command waits for a trigger.
Not available in SCPI trigger.digout[N].assert() (on page 14-452)	This command asserts a trigger pulse on one of the digital I/O lines.
:TRIGger:DIGital<n>:OUT:LOGic (on page 12-305) trigger.digout[N].logic (on page 14-453)	This command sets the output logic of the trigger event generator to positive or negative for the specified line.

SCPI command TSP command	Description
:TRIGger:DIGital<n>:OUT:PULSewidth (on page 12-306) trigger.digout[N].pulsewidth (on page 14-454)	This command describes the length of time that the trigger line is asserted for output triggers.
Not available in SCPI trigger.digout[N].release() (on page 14-454)	This command releases an indefinite length or latched trigger.
:TRIGger:DIGital<n>:OUT:STIMulus (on page 12-306) trigger.digout[N].stimulus (on page 14-455)	This command selects the event that causes a trigger to be asserted on the digital output line.

NOTE

To use the trigger model as a stimulus to a digital I/O line, you can use the trigger model Notify block. For information on the Notify block, see Notify block.

Digital I/O bit weighting

Bit weighting for the digital I/O lines is shown in the following table. Line 1 is the least significant bit.

Line #	Bit	Pin	Decimal	Hexadecimal	Binary
1	B1	1	1	0x01	000001
2	B2	2	2	0x02	000010
3	B3	3	4	0x04	000100
4	B4	4	8	0x08	001000
5	B5	6	16	0x10	010000
6	B6	8	32	0x20	100000

Digital I/O programming examples

These examples provide typical methods you can use to work with the digital I/O port.

Outputting a bit pattern

The programming examples below illustrate how to output the bit pattern 110101 at the digital I/O port. Line 1 (bit 1) is the least significant bit.

Using SCPI commands to configure all six lines as digital outputs:

```
:DIGital:LINE1:MODE DIGital, OUT
:DIGital:LINE2:MODE DIGital, OUT
:DIGital:LINE3:MODE DIGital, OUT
:DIGital:LINE4:MODE DIGital, OUT
:DIGital:LINE5:MODE DIGital, OUT
:DIGital:LINE6:MODE DIGital, OUT
```

Using SCPI commands to set the state of each line individually:

```
:DIGital:LINE6:STATe 1
:DIGital:LINE5:STATe 1
:DIGital:LINE4:STATe 0
:DIGital:LINE3:STATe 1
:DIGital:LINE2:STATe 0
:DIGital:LINE1:STATe 1
```

Using SCPI commands to set all six lines at once by writing the decimal equivalent of the bit pattern to the port:

```
:DIGital:WRITe 53
```

Using TSP commands to configure all six lines as digital outputs:

```
-- Send for loop as a single chunk or include in a script.
for i = 1, 6 do
    digio.line[i].mode = digio.MODE_DIGITAL_OUT
end
```

Using TSP commands to set the state of each line individually:

```
digio.line[1].state = digio.STATE_HIGH
digio.line[2].state = digio.STATE_LOW
digio.line[3].state = digio.STATE_HIGH
-- You can use 0 instead of digio.STATE_LOW.
digio.line[4].state = 0
-- You can use 1 instead of digio.STATE_HIGH.
digio.line[5].state = 1
digio.line[6].state = 1
```

Using TSP commands to set all six lines at once by writing the decimal equivalent of the bit pattern to the port:

```
-- You can write binary, decimal or hexadecimal values, as shown below.
-- Use binary value.
digio.writeport(0b110101)
-- Use decimal value.
digio.writeport(53)
-- Use hexadecimal value.
digio.writeport(0x35)
```

Reading a bit pattern

The programming examples below illustrate how to read part or all of a bit pattern that has been applied to the digital I/O port by an external instrument. The binary pattern is 111111 (63 decimal). Line 1 (bit 1) is the least significant bit.

Using SCPI commands:

Configure all six lines as digital inputs:

```
DIGital:LINE1:MODE DIGital, IN  
DIGital:LINE2:MODE DIGital, IN  
DIGital:LINE3:MODE DIGital, IN  
DIGital:LINE4:MODE DIGital, IN  
DIGital:LINE5:MODE DIGital, IN  
DIGital:LINE6:MODE DIGital, IN
```

Read the state of Line 2:

```
DIGital:LINE2:STATe?
```

Value returned is 1.

Read the state of Line 3:

```
DIGital:LINE3:STATe?
```

Value returned is 1.

Read the value applied to the entire port:

```
DIGital:READ?
```

Value returned is 63, which is the decimal equivalent of the binary bit pattern.

Using TSP commands:

```
-- Configure all six digital I/O lines as digital inputs.
-- You can also use a for loop.
digio.line[1].mode = digio.MODE_DIGITAL_IN
digio.line[2].mode = digio.MODE_DIGITAL_IN
digio.line[3].mode = digio.MODE_DIGITAL_IN
digio.line[4].mode = digio.MODE_DIGITAL_IN
digio.line[5].mode = digio.MODE_DIGITAL_IN
digio.line[6].mode = digio.MODE_DIGITAL_IN
-- Read and then print the state of Line 2 (bit 2).
b2 = digio.line[2].state
print(b2)
```

The value returned is `digio.STATE_HIGH`.

```
-- Print the state of Line 3 (bit 3).
print(digio.line[3].state)
```

The value returned is `digio.STATE_HIGH`.

```
-- Read and then print the value applied to the entire port.
port = digio.readport()
print(port)
```

The value returned is 63, which is the decimal equivalent of the binary bit pattern.

External trigger control

You can use the EXTERNAL TRIGGER IN and EXTERNAL TRIGGER OUT terminals on the rear panel of the DAQ6510 to initiate an action on the instrument or on another instrument.

EXTERNAL TRIGGER OUT is TTL-compatible output line with a 0 V to 5 V logic signal. The instrument can generate output trigger pulses on this line. You can use this line for triggering by using the transition of the line state to initiate an action on an instrument monitoring this line. The connector is a BNC type.

EXTERNAL TRIGGER IN is a TTL-compatible input line with a 0 V to 5 V logic signal. You can trigger the DAQ6510 by using the transition of the line state by another device to initiate an action. The instrument can detect input trigger pulses on this line. The connector is a BNC type.

Setting up the external I/O

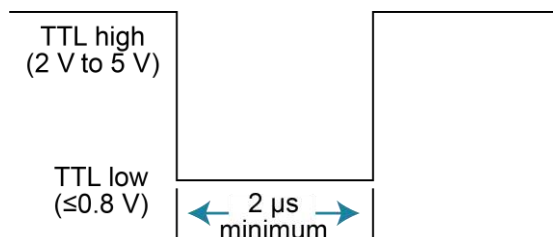
You cannot configure or directly control the EXTERNAL TRIGGER IN and EXTERNAL TRIGGER OUT lines from the front panel. To configure and control them, you need to send commands to the DAQ6510 over a remote interface. You can use either the SCPI or TSP command set. See [Remote communications interfaces](#) (on page 2-7) for information about setting up a remote interface and choosing a command set.

The options to set up the external I/O are stimulus, edge, and logic.

The stimulus selects the event that causes a trigger to be asserted on the EXTERNAL TRIGGER OUT line. You can use any of the standard trigger events with the external I/O. See [Trigger events](#) (on page 8-69) for a list of the available trigger events.

The edge sets the type of edge that is detected as an input on the EXTERNAL TRIGGER IN trigger line. You can set the DAQ6510 to detect trigger inputs on the falling edge, rising edge, or either edge. When falling edge is selected, the input is detected when the line state transitions from high to low. When rising edge is selected, the input is detected when the line state transitions from low to high. The following figure shows the electrical and timing specifications for pulse detection of external trigger in.

Figure 145: External trigger in pulse specifications



The logic type determines if the output asserts a TTL-high pulse or a TTL-low pulse for the trigger.

You can use the EXTERNAL TRIGGER IN and EXTERNAL TRIGGER OUT lines with interactive triggering or with the trigger model. For more information about the trigger modes and triggering, refer to [Triggering](#) (on page 8-18).

Remote external I/O commands

Commands for both SCPI and TSP are summarized in the following table.

SCPI command TSP command	Description
A line reset is not available in SCPI; however, the line is reset when a global reset (*RST) is sent trigger.ext.reset() (on page 14-458)	This command resets the edge, logic, and stimulus values for the EXTERNAL TRIGGER IN and EXTERNAL TRIGGER OUT lines to their default values.
:TRIGger:EXTErnal:IN:CLEar (on page 12-308) trigger.extin.clear() (on page 14-459)	This command clears the trigger event on the EXTERNAL TRIGGER IN line.
:TRIGger:EXTErnal:IN:EDGE (on page 12-309) trigger.extin.edge (on page 14-460)	This command sets the type of edge that is detected as an input on the EXTERNAL TRIGGER IN trigger line.
:TRIGger:EXTErnal:IN:OVERrun? (on page 12-310) trigger.extin.override (on page 14-460)	This command returns the event detector overrun status.
Not available in SCPI trigger.extin.wait() (on page 14-461)	This command waits for an input trigger.
Not available in SCPI trigger.extout.assert() (on page 14-462)	This command asserts a trigger on the external I/O line.
:TRIGger:EXTErnal:OUT:LOGic (on page 12-311) trigger.extout.logic (on page 14-462)	This command sets the output logic of the trigger event generator to positive or negative for the EXTERNAL TRIGGER OUT line.
:TRIGger:EXTErnal:OUT:STIMulus (on page 12-312) trigger.extout.stimulus (on page 14-463)	This command selects the event that causes a trigger to be asserted on the EXTERNAL TRIGGER OUT line.

NOTE

To use the trigger model as a stimulus to the external I/O line, you can use the trigger model Notify block. For information, see Notify block.

Triggering

Triggering allows you to start and synchronize scan start and scan step operations, and measure actions on one or more instruments with a trigger event or a combination of trigger events that you set. This section describes some of the options available for triggering, including command interface triggering, timers, analog trigger, and event blenders.

Command interface triggering

A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 `device_trigger` method is invoked (VXI-11 only)
- A USBTMC trigger message is received (USB only)
- A `*TRG` message is received

To use a command interface trigger event as an input stimulus for another trigger object, set the stimulus as TSP event `trigger.EVENT_COMMAND` or the SCPI event `COMMAND`. To ensure that trigger commands that are issued over the command interface are processed in the correct order, the instrument does not generate a trigger event until:

- The trigger command is executed
- TSP only: `trigger.wait()` retrieves the trigger command from the command queue before it would normally be executed

Command interface triggering does not generate action overruns. The triggers are processed in the order that they are received in the DAQ6510 command queue. The DAQ6510 only processes incoming commands when no commands are running. Unprocessed input triggers can cause an overflow in the command queue. It is important to make sure a script processes triggers while it is running.

NOTE

If you are using a test script using TSP, the command queue can fill up with trigger entries if over 50 `*TRG` messages are received while a test script is running, even if the script is processing triggers. You can avoid this by using the [localnode.prompts4882](#) (on page 14-360) attribute, and by using `trigger.wait()` calls that remove the `*TRG` messages from the command queue. If the command queue fills with too many trigger entries, messages such as `abort` are not processed.

Triggering using hardware lines

NOTE

The digital I/O and TSP-Link options require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

You can use the digital I/O lines, rear-panel EXTERNAL TRIGGER OUT and EXTERNAL TRIGGER IN lines, and TSP-Link[®] synchronization lines to synchronize the operations of the DAQ6510 with those of external instruments. You can use these lines to synchronize the DAQ6510 with other TSP-enabled instruments, including other DAQ6510 instruments. You must use the digital I/O lines or the EXTERNAL TRIGGER OUT and EXTERNAL TRIGGER IN lines to synchronize the DAQ6510 with other Keithley products or other non-Keithley products.

The lines are configured and controlled similarly. See [Digital I/O](#) (on page 8-1), [TSP-Link System Expansion Interface](#) (on page 9-1), or [External trigger control](#) (on page 8-16) for information about connections and configuration and control of the lines.

Analog triggering overview

You can use input signals for triggering when you are measuring current or voltage using the DC measure or digitize functions. The instrument uses the measurements to determine if the trigger condition has been met. The trigger occurs when the signal satisfies the specified conditions. Triggers generated by these comparisons are called analog triggers. You can use analog triggers to trigger instrument action in the same ways that you use other trigger types.

Analog triggers are available for channels using remote commands. You cannot set up analog triggers for channels using the front panel.

The DAQ6510 analog trigger has the same DC accuracy as normal readings.

Analog triggers should be set after other instrument settings. For example, before you set up analog triggers, select the measurement function, turn autozero off, and select a specific range.

Analog trigger mode

To set up an analog trigger, you need to define the analog trigger mode. The mode defines how the instrument processes the signal that generates the trigger event. The available modes are edge and window.

When edge is selected, the trigger event occurs when the signal crosses a level that you define. You also specify if the trigger occurs on the rising or falling edge of the signal.

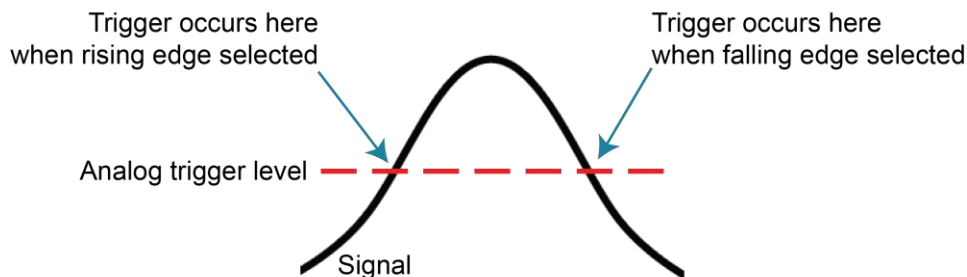
The window mode is typically used to spot signal anomalies.

When window is selected, the trigger event occurs when the signal enters or exits a window that is defined by low and high signal levels.

Edge mode

Edge triggers occur when you cross a defined signal level. When you select the edge trigger mode, you also need to set the trigger level and the slope (rising or falling). Refer to the following figure for an example signal.

Figure 146: Edge analog trigger mode

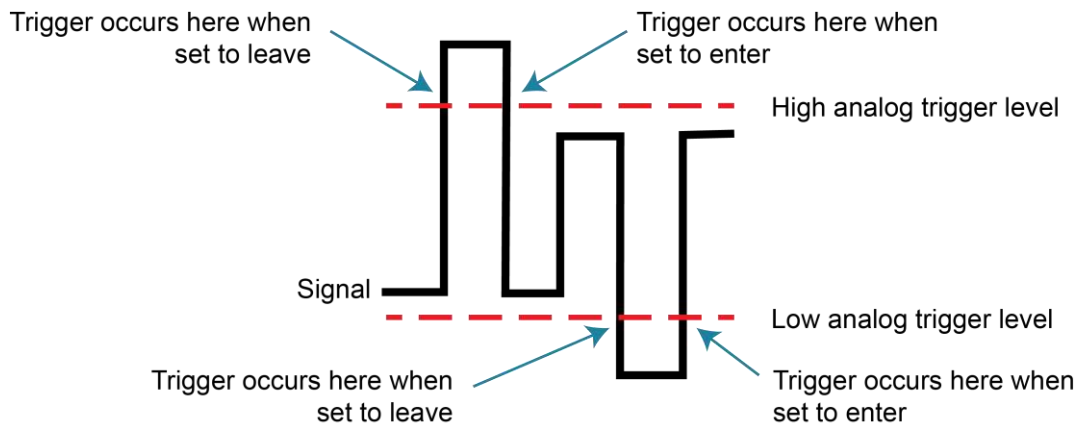


Window mode

Window triggers occur when a signal enters or leaves a defined signal window.

When you set up window mode, you define the high and low analog trigger levels. You also define whether the trigger should occur when the signal enters or leaves the window.

Figure 147: Window analog trigger mode



Analog triggering example with digitize function

This example shows how to set up the analog trigger for a digitize function. The trigger is used to pulse EXTERNAL TRIGGER OUT. Set the instrument to use the front terminals before running these examples.

When the script runs, it opens the Graph tab on the front panel. To view the data, open the Scale tab and set X-Axis Method to Show Group of Readings.

Analog trigger example – front panel

An example of how to use the analog trigger options from the front panel:

1. Select the **FUNCTION** key and select **Digitize Functions**, then **Digitize Voltage**.
2. Select the **HOME** key.
3. Swipe to the **Settings** screen.
4. Set the Sample Rate to **100000**.
5. Select **Set Up Trigger**.
6. Set the Source Event to **Waveform**.
7. Select **Analog Edge**.
8. Set the Position to **25%**.
9. Set the level to **0.5V**.
10. Select the **Graph** tab.
11. Press the **TRIGGER** key.

Analog trigger example – SCPI

The DMM is set to digitize voltage with a sample rate of 100,000 samples per second.

The analog trigger mode is set to detect an edge with a level of 0.5 V.

Set the reading buffer size to 100,000.

Set up the Loop Until Event trigger model to make continuous measurements until the analog edge trigger event occurs, keeping up to 25,000 pre-trigger measurements, followed by 75,000 post-trigger measurements.

Open the graph screen.

```
*RST
:SENSe:DIGitize:FUNctIon "VOLT"
:SENSe:DIGitize:VOLTagE:SRATe 100000
:DIG:VOLT:ATR:MODE EDGE
:DIG:VOLT:ATR:EDGE:LEV 0.5
:TRACe:POINts 100000, "defbuffer1"
:TRIGger:LOAD "LoopUntilEvent", ATRigger, 25, ENTer, 0, "defbuffer1"
:DISPlay:SCReen GRAPh
*WAI
:INIT
```

Analog trigger example – TSP

```
-- Set the instrument to the default settings.
reset()
-- Set the Notify 2 event to trigger the TRIGGER OUT line.
trigger.extout.stimulus = trigger.EVENT_NOTIFY2

-- Set up the DMM.
-- Select the digitize voltage function.
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
-- Set the sample rate to 100,000 samples per second
dmm.digitize.samplerate = 100000
-- Set analog trigger mode to edge detect with a level of 0.5 V.
dmm.digitize.analogtrigger.mode = dmm.MODE_EDGE
dmm.digitize.analogtrigger.edge.level = 0.5
-- Set count to 1000 readings per trigger condition
dmm.digitize.count = 1000

-- Set reading buffer size to 100,000.
defbuffer1.capacity = 100000

-- Set up the Loop Until Event trigger model to make continuous measurements
-- until the analog edge trigger event occurs, making 25,000 measurements.
trigger.model.load("LoopUntilEvent", trigger.EVENT_ANALOGTRIGGER, 25,
    trigger.CLEAR_ENTER, 0, defbuffer1)
-- Open the graph screen
display.changescreen(display.SCREEN_GRAPH)
waitcomplete()
-- Start the trigger model.
trigger.model.initiate()
```

LAN triggering overview

You can send and receive triggers over the LAN interface. The DAQ6510 supports LAN extensions for instrumentation (LXI). It has eight LAN triggers that generate and respond to LXI trigger packets.

Understanding hardware value and pseudo line state

LAN triggering and hardware synchronization are similar, except that LAN triggering uses LXI trigger packets instead of hardware signals. A bit in the LXI trigger packet called the hardware value simulates the state of a hardware trigger line. The DAQ6510 stores the hardware value as the pseudo-line state. Only the state of the last LXI trigger packet that was sent or received is stored.

The stateless event flag is a bit in the LXI trigger packet that indicates if the hardware value should be ignored. If it is set, the DAQ6510 ignores the hardware value of the packet and generates a trigger event. The DAQ6510 always sets the stateless flag for outgoing LXI trigger packets. If the stateless event flag is not set, the hardware value indicates the state of the signal.

The instrument interprets changes in the hardware value of consecutive LXI trigger packets as edge transitions. Edge transitions generate trigger events. If the hardware value does not change between successive LXI trigger packets, the DAQ6510 assumes an edge transition was missed and generates a trigger event. The following table shows edge detection in LAN triggering.

LXI trigger edge detection

Stateless event flag	Hardware value	Pseudo-line state	Falling edge	Rising edge
0	0	0	Detected	Detected
0	1	0	-	Detected
0	0	1	Detected	-
0	1	1	Detected	Detected
1	-	-	Detected	Detected

You can set the LAN trigger edge detection method in incoming LXI trigger packets. The edge that is selected also determines the hardware value in outgoing LXI trigger packets. The following table lists the LAN trigger edges.

Trigger mode	Input detected	Output generated
Either edge	Either	Negative
Falling edge	Falling	Negative
Rising edge	Rising	Positive

LAN trigger objects generate LXI trigger events, which are LAN0 to LAN7 (zero based). To specify the LAN trigger event in a command, use `LANN`, where *N* is 1 to 8. `LAN1` corresponds to LXI trigger event LAN0 and `LAN8` corresponds to LXI trigger event LAN7. To specify the LAN trigger event in a command, see [Trigger events](#) (on page 8-69).

Generate LXI trigger packets

You can configure the DAQ6510 to output an LXI trigger packet to other LXI instruments.

To generate LXI trigger packets:

1. Call the SCPI `:TRIGger:LAN<n>:OUT:CONNeCT:STATe` command or TSP `trigger.lanout[N].connect()` function.
2. Select the event that triggers the outgoing LXI trigger packet by assigning the specific event to the LAN stimulus input using the SCPI `:TRIGger:LAN<n>:OUT:STIMulus` command or TSP `trigger.lanout[N].stimulus` attribute.

Make sure to use the same LXI domain on both the DAQ6510 instrument and the other instrument. If the DAQ6510 has a different LXI domain from the instrument at the other end of the trigger connection, the LXI trigger packets are ignored by both instruments.

Trigger timers

You can assign trigger timers as events that trigger a scan to start, make a measurement, or start a channel action. You can also assign them in trigger model wait, branch on event, and notify blocks. If you are using remote commands, you can set the trigger timers to be the stimulus for the EXTERNAL TRIGGER OUT, LAN, blender, or TSP-Link output.

The DAQ6510 has four independent timers.

You can set the count, delay, and time when the trigger occurs for the trigger timers. You need to set these parameters before you enable the trigger timers.

Count

The count sets the number of events to generate each time the timer generates a trigger event. Each event is separated by the trigger timer delay.

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

If you are using remote commands, the delay is set by the SCPI

:TRIGger:TIMer<n>:DElay or TSP `trigger.timer[N].delay` command. The count is set using the SCPI command :TRIGger:TIMer<n>:COUNT or the TSP command `trigger.timer[N].count`.

Timer delays

You can set up the timers to perform delays. A delay is the period after the timer is triggered and before the timer generates a trigger event. All delay values are specified in seconds.

Delay lists, which are only available using remote TSP commands, allow the timer to sequence through an array of delay values. Delay lists allow the timer to use a different interval each time it performs a delay. Each time the timer generates a trigger event, it uses the next delay in the list. The timer repeats the delay list after all the elements in the delay list have been used.

If you use the trigger timer with a trigger model, make sure the trigger timer delay is set so that the readings are paced correctly.

Using SCPI commands:

To set up a 50 μ s trigger timer delay for timer 2, send the command:

```
TRIGger:TIMer2:DElay 50E-6
```

Using TSP commands:

To set up a 50 μ s trigger timer delay for timer 2, send the command:

```
trigger.timer[2].delay = 50e-6
```

To set up a delay list for timer 3 for delays of 2, 10, 15, and 7 s, send the command:

```
trigger.timer[3].delaylist = {2, 10, 15, 7}
```

Define when to generate a timer event

You can specify if the trigger timer starts when an event occurs or at a specific time.

On the front panel, the timer Start Condition option sets when the timer will start. If you select Trigger Event, the trigger timer starts when the event defined in Stimulus occurs. If you select Specific Date and Time, you can select the date and time when the trigger occurs.

If you are using remote comments, you can specify when timer events are generated using the SCPI `:TRIGger:TIMer<n>:STARt:GENerate` or TSP `trigger.timer[N].start.generate` command. When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses.

You can watch for a stimulus before starting the timer by using the SCPI `:TRIGger:TIMer<n>:STARt:STIMulus` command or `trigger.timer[N].start.stimulus` command.

The following example resets trigger timer 4 (TSP only), then sets it to a 0.5 s delay, a stimulus of notify 5, to occur when the timer delay elapses, and a count of 20, and enables the timer.

SCPI example

```
TRIG:TIM4:DEL 0.5
TRIG:TIM4:STAR:STIM NOTIfy5
TRIG:TIM4:STAR:GEN ON OFF
TRIG:TIM4:COUN 20
TRIG:TIM4:STAT ON
```

TSP example

```
trigger.timer[4].reset()
trigger.timer[4].delay = 0.5
trigger.timer[4].start.stimulus = trigger.EVENT_NOTIFY5
trigger.timer[4].start.generate = trigger.OFF
trigger.timer[4].count = 20
trigger.timer[4].enable = trigger.ON
```

You can also set a time when the timer will start using the seconds and fractional seconds commands. (SCPI commands `:TRIGger:TIMer<n>:STARt:SEConds` and `:TRIGger:TIMer<n>:STARt:FRACTIONal`; TSP commands `trigger.timer[N].start.seconds` and `trigger.timer[N].start.fractionalseconds`.) When you specify a time, the timer starts immediately if the time has passed.

Timer action overruns

The timer receives an action overrun when it generates a trigger event while a timer delay is still in progress. Use the status model to monitor for the occurrence of action overruns. For details, see [Status model](#) (on page 16-1).

Using trigger timers with timing blocks

For precise timing or if you need to synchronize timing with other execution blocks or events, you can use the trigger timer commands with trigger model wait blocks and notify blocks. You can use the trigger timer commands to add small precise delays or to start measurements or to overcome variable measurement delays. The DAQ6510 has 1 to 4 independent timers.

For example, you can use a trigger timer to control the delay between nonsequential blocks. After creating a trigger timer, you can insert a notify block to start the timer at a specific point in the trigger model. You could then add a wait block to wait for the timer to expire.

Another example is a measure/digitize block that takes a variable amount of time. To ensure a precise time between measurements, you can create a trigger timer and define it to be a fixed interval that is longer than the longest possible measurement. Then you can set up the trigger model to include:

- A notify block that starts the trigger timer
- A measure/digitize block that makes a measurement
- A wait block that waits for the timer to expire
- A branch counter block that iterates some number of times

NOTE

Some attributes of trigger timers should not be used with the trigger model. Attributes you should not set are:

- Count value of 0 (resulting in generation of trigger events indefinitely)
 - Delay lists
-

Remote trigger timer commands

SCPI trigger timer commands:

- [:TRIGger:TIMer<n>:CLEar](#) (on page 12-335)
- [:TRIGger:TIMer<n>:COUNT](#) (on page 12-335)
- [:TRIGger:TIMer<n>:DELay](#) (on page 12-337)
- [:TRIGger:TIMer<n>:START:FRACTIONal](#) (on page 12-337)
- [:TRIGger:TIMer<n>:START:GENerate](#) (on page 12-338)
- [:TRIGger:TIMer<n>:START:OVERrun?](#) (on page 12-339)
- [:TRIGger:TIMer<n>:START:SEConds](#) (on page 12-339)
- [:TRIGger:TIMer<n>:START:STIMulus](#) (on page 12-340)
- [:TRIGger:TIMer<n>:STATe](#) (on page 12-342)

TSP trigger timer commands:

- [trigger.timer\[N\].clear\(\)](#) (on page 14-521)
- [trigger.timer\[N\].count](#) (on page 14-522)
- [trigger.timer\[N\].delay](#) (on page 14-523)
- [trigger.timer\[N\].delaylist](#) (on page 14-524)
- [trigger.timer\[N\].enable](#) (on page 14-525)
- [trigger.timer\[N\].reset\(\)](#) (on page 14-526)
- [trigger.timer\[N\].start.fractionalseconds](#) (on page 14-527)
- [trigger.timer\[N\].start.generate](#) (on page 14-528)
- [trigger.timer\[N\].start.overrun](#) (on page 14-529)
- [trigger.timer\[N\].start.seconds](#) (on page 14-530)
- [trigger.timer\[N\].start.stimulus](#) (on page 14-530)
- [trigger.timer\[N\].wait\(\)](#) (on page 14-533)

Event blenders

The ability to combine trigger events is called event blending. You can use an event blender to wait for up to four input trigger events to occur before responding with an output event.

You set the event blender operation using remote commands. You cannot set them up through the front panel.

You can program up to two event blenders for the DAQ6510.

Event blender operations

You can use event blenders to perform logical AND or logical OR operations on trigger events. For example, trigger events can be triggered when either a manual trigger or external input trigger is detected.

When AND operation is selected, the event blender generates an event when an event is detected on all the assigned stimulus inputs.

When OR operation is selected, the event blender generates an event when an event is detected on any one of the four stimulus inputs.

Using SCPI commands:

Send the command `:TRIGger:BLENDER<n>:MODE`.

Set the command to `OR` or `AND`.

Using TSP commands:

Send the command `trigger.blender[N].orenable`.

Setting the command to `true` enables OR operation; setting it to `false` enables AND operation.

Assigning blender trigger events

Each event blender has four stimulus inputs. You can assign a different trigger event to each stimulus input.

You set the blender stimulus events using remote commands. See the command descriptions for the list of events that you can assign.

Using SCPI commands:

Send the command `:TRIGger:BLENDER<n>:STIMulus<m>`.

Using TSP commands:

Send the command `trigger.blender[N].stimulus[M]`.

Trigger blender action overruns

The event blenders can generate action overruns.

When the event blender operation is set to AND, overruns occur when a second event on any of its inputs is detected before an output event is generated.

When the operation is set to OR, overruns occur when two events are detected simultaneously.

Use the status model to monitor for the occurrence of action overruns. For details, see [Status model](#) (on page 16-1).

Interactive triggering

NOTE

Some of the following examples require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Interactive triggering is only available if you are using the TSP command set.

If you need more control of triggering than you can get using a trigger model, you can use interactive triggering to enable your system to generate and detect trigger events anywhere in the test flow. Interactive triggering is typically used in the context of TSP script operation. For example, interactive triggering can be used when you need to implement conditional branching to other test setups based on recent measurements.

All the DAQ6510 trigger objects have built-in event detectors that monitor for trigger events. The event detector only monitors events generated by that object. They cannot be configured to monitor events generated by any other trigger object.

You can use the `wait()` function of the trigger object to cause the instrument to suspend command execution until a trigger event occurs or until the specified timeout period elapses. For example, use `trigger.blender[N].wait(timeout)` to suspend command execution until an event blender generates an event, where *N* is the specific event blender and *timeout* is the timeout period. After executing the `wait()` function, the event detector of the trigger object is cleared.

The following programming example illustrates how to suspend command execution while waiting for various events to occur:

```
-- Wait up to 60 seconds for timer 1 to complete its delay.
trigger.timer[1].wait(60)
-- Wait up to 30 seconds for input trigger to digital I/O line 5.
trigger.digin[5].wait(30)
```

You can use some trigger objects to generate output triggers on demand. These trigger objects are the external I/O line, digital I/O lines, the TSP-Link synchronization lines, and the LAN.

The programming example below generates output triggers using the `assert` function of the trigger object.

```
-- Generate a 20 us pulse on digital I/O line 3.
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.digout[3].pulsewidth = 20e-6
trigger.digout[3].assert()
-- Generate a rising edge trigger on TSP-Link sync line 1.
tsplink.line[1].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkin[1].edge = trigger.EDGE_RISING
trigger.tsplinkout[1].logic = trigger.LOGIC_POSITIVE
trigger.tsplinkout[1].assert()
-- Generate a LAN trigger on LAN pseudo line 6.
-- Note that connection parameters and commands that
-- establish a connection are not shown.
trigger.lanout[6].assert()
```

Use the release function to allow the hardware line to output another external trigger when the pulse width is set to 0.

Setting the pulse width to 0 results in an indefinite length pulse when the assert function is used to output an external trigger. When an indefinite length pulse is used, the release function must be used to release the line before another external trigger can be output.

The release function can also be used to release latched input triggers when the hardware line mode is set to synchronous. In synchronous mode, the receipt of a falling edge trigger latches the line low. The release function releases this line high in preparation for another input trigger.

The programming example below illustrates how to output an indefinite external trigger.

```
-- Set digio line 1 to output an indefinite external trigger.
digio.line[1].mode = digio.MODE_TRIGGER_OUT
trigger.digout[1].logic = trigger.LOGIC_NEGATIVE
trigger.digout[1].pulsewidth = 0
trigger.digout[1].assert()
-- Release digio line 1.
trigger.digout[1].release()
-- Output another external trigger.
trigger.digout[1].assert()
```

For information about hardware lines, see [Digital I/O lines](#) (on page 8-5), [External I/O](#) (on page 8-16), and [Triggering using TSP-Link trigger lines](#) (on page 9-8).

The programming example below checks and responds to detector overruns.

```
testOver = trigger.digin[4].overrun
if testOver == true then
    print("Digital I/O overrun occurred.")
end
```

Trigger model

The trigger model controls the sequence in which measure actions occur. The DAQ6510 trigger model is flexible, allowing you to control as much or as little as needed for your measurement application.

When you are setting up a trigger model, you can choose the following options:

- Wait for an event to occur before making another measurement
- Notify other equipment and timers that an event has occurred
- Wait for another piece of equipment to signal completion
- Use measure configuration lists to apply different measure settings dynamically during trigger model operation
- Specify delays between events and measurements
- Store measurements into a given buffer until an event occurs, then switch to another buffer
- Conditionally take actions based on whether the measurement falls within set limits

Additional options are detailed in the following sections.

The DAQ6510 includes trigger-model templates to allow you to quickly implement a trigger model. You can also set up your own trigger models.

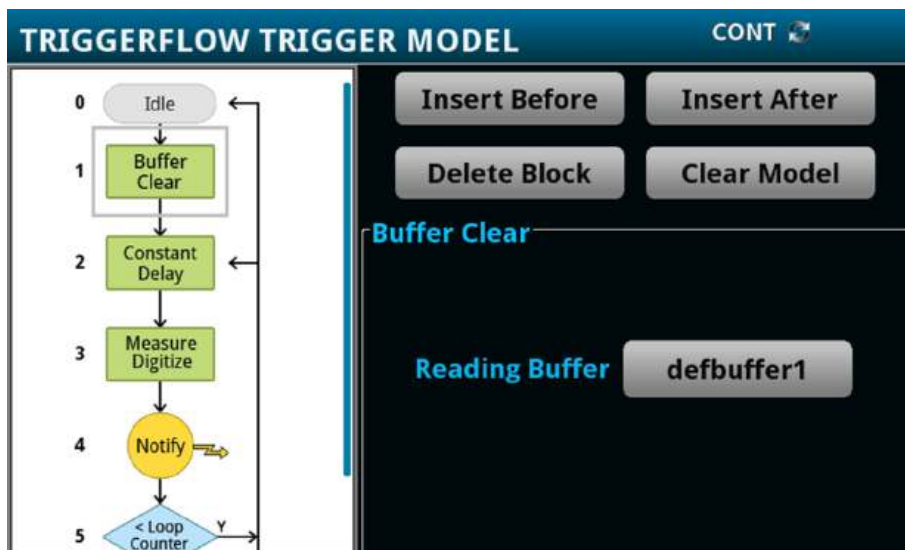
NOTE

Be sure to save any existing trigger models or scans before creating a new trigger model or scan. Scans are implemented by using the trigger model. When you create a scan, a new trigger model is created. Any existing trigger models are removed. Conversely, creating a new trigger model removes the scan. For information on saving scans and trigger models, refer to [Saving setups](#) (on page 4-92).

TriggerFlow Trigger Model

The TriggerFlow® Trigger Model diagram on the front panel provides an interactive visual flowchart of the trigger model. An example of the TriggerFlow diagram when the SimpleLoop trigger template is selected is shown in the following figure.

Figure 148: TriggerFlow Trigger Model screen



You can swipe the TriggerFlow diagram to view the entire trigger model. You can also insert, delete, and adjust settings for the blocks in the trigger model. Block numbers and branching paths are automatically adjusted when you insert or delete blocks.

To insert a block, select a block in the TriggerFlow diagram. Select **Insert Before** to insert a block immediately above the selected block. Select **Insert After** to insert a block immediately below the selected block.

To delete block, select a block the TriggerFlow diagram and select **Delete Block**.

To remove all blocks from the TriggerFlow diagram, select **Clear Model**.

To change the settings for a trigger block, select the block. The settings that are available for that block are displayed on the right side of the screen. Refer to the descriptions of the blocks in the [Trigger model blocks](#) (on page 8-37) for detail on the options that are available for each block.

Trigger-model blocks

Each trigger model consists of blocks that can be combined to create the trigger model. The blocks can be combined from the front panel or by sending remote commands. You can connect a maximum of 63 blocks as needed to control the instrument.

You can combine trigger-model blocks as you would construct a flowchart diagram. Trigger models are created using these fundamental blocks:

- **Wait:** Waits for an event to occur before the flow continues
- **Action:** Starts an action in the instrument, such as making a measurement or clearing a buffer
- **Notify:** Notifies other equipment or timers that an event has occurred
- **Branch:** Branches when a condition has been satisfied

NOTE

If you set up a scan, Channel Action, Start, and End blocks are added to the TriggerFlow diagram. These blocks are set up internally as part of the scan and cannot be changed using commands or the front panel.

Each type of block is described in the following topics.

Wait block

The wait block causes the trigger model to stop and wait for an event or set of events to occur before continuing. You can specify up to three events for each wait block.

The event can occur before the trigger model reaches the wait block. If the event occurs after the trigger model starts but before the trigger model reaches the wait block, the trigger model records the event. By default, when the trigger model reaches the wait block, it executes the wait block without waiting for the event to happen again (the clear parameter is set to never).

The instrument clears the memory of the recorded event when the trigger model exits the wait block. It also clears the recorded trigger event when the clear parameter is set to enter.

All items in the list are subject to the same action. You cannot combine **AND** and **OR** logic in a single wait block.

When you select the wait block, the following options are available.

Setting	Description
Clear	To clear previously detected trigger events when entering the wait block, select Enter. To immediately act on any previously detected triggers and not clear them, select Never.
Event 1	An event that must occur before the trigger block will continue.
Event 2	Optional. An event that must occur before the trigger block will continue.
Event 3	Optional. An event that must occur before the trigger block will continue.
Event Logic	Optional. Determines if all the defined events must occur or if at least one of the events must occur. Select AND if all the defined events must occur. Select OR if at least one of the events must occur.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

The event can be any of the events described in the following table.

Event	Description
Analog Trigger	Use the analog trigger.
Blender	Wait for the events set by an event blender.
Command	A command interface trigger (bus trigger): ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command <code>device_trigger</code>
Digital Input	Line edge detected on a digital input line. When you select this option, you select the digital input to monitor. After selecting the digital input line, choose Settings to select the type of edge (falling, rising, or either).
Display TRIGGER Key	Front-panel TRIGGER key press.
External Trigger In	Use a pulse from the external trigger. When you select this option, choose Settings to select the type of edge (falling, rising, or either).
LAN Input Trigger	An LXI trigger packet is received on the LAN trigger object. When you select this option, you select the LAN trigger to monitor. After you select the line, choose Settings to select the type of edge (falling, rising, or either).
None	No trigger event.
Timer	A trigger timer expired. When you select this option, you select the timer to monitor. After selecting the timer, choose Settings to define the timer. Refer to Trigger timers (on page 8-26) for detail on the options. For a timer to expire, you must start it. One method to start the timer in the Trigger Model is to include a Notify block before this block. Set the Notify block to use the same timer.
TSP-Link Input	Line edge detected on a TSP-Link synchronization line. When you select this option, you select the TSP-Link line to monitor. After selecting the TSP-Link line, choose Settings to select the type of edge (falling, rising, or either).

If you need to set up the trigger model to wait for an event under some conditions but not others, you can use a branch block. For information, see [Branching blocks](#) (on page 8-47).

Action blocks

The action blocks start an action in the instrument, such as making a measurement or clearing a buffer.

Measure/Digitize block

This block triggers measurements based on the measure function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

1. The instrument begins triggering measurements.
2. The trigger model execution waits for the measurement to be made.
3. The instrument processes the reading and places it into the specified reading buffer.

If you are defining a user-defined reading buffer, you must create it before you define this block.

When you set the count to a finite value, trigger model execution does not proceed until all operations are complete.

If you set the count to infinite, the trigger model executes subsequent blocks when the measurement is made; the triggering of measurements continues in the background until the trigger model execution reaches another measure/digitize block or until the trigger model ends. To use infinite, there must be a block after the measure/digitize block in the trigger model, such as a wait block. If there is no subsequent block, the trigger model stops, which stops measurements.

When you set the count to auto, the trigger model uses the count value that is active for the selected function instead of a specific value. You can use this with configuration lists to change the count value each time a measure/digitize block is encountered.

When the function is set to digitize, there is a 2 μ s delay after the block makes the last measurement in the count. For example, if there are two readings at a sample rate of 20,000 samples per second (50 μ s apart) with an aperture of 1 μ s with a delay of 100 μ s, the delay starts at 51 μ s. The first reading occurs at 0 μ s and the second starts at 50 μ s, but it is completed at 51 μ s because the aperture is only 1 μ s. If the aperture is set to Auto, the first reading is at 0 μ s, the second starts at 50 μ s, and the delay starts at 100 μ s.

A trigger model that digitizes measurements may appear to hang in the wait block because it is making many measurements in one block.

NOTE

Earlier firmware versions of the DAQ6510 had separate measure and digitize blocks. If you have code that is using those blocks, they work in this version of the DAQ6510 firmware. The digitize block cannot be used to digitize measurements on an instrument that does not have the digitize feature.

NOTE

If you bring in code that uses a measure or digitize block and does not define the count, the count is set to 1.

When you select the Measure/Digitize block, the following options are available.

Setting	Description
Count	Specifies the number of readings to make before moving to the next block in the trigger model. You can select: ■ A specific number ■ Infinite: Run continuously until stopped. ■ Stop Infinite: Stop the infinite setting. ■ Auto: Use the measure count setting that is active for the function for the trigger model block.
Reading Buffer	The name of the buffer where the readings are stored.

Constant delay block

When trigger model execution reaches a delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made, and if any previously executed block started infinite measurements, they also continue to be made.

This delay waits for the delay time to elapse before proceeding to the next block in the trigger model.

If other delays have been set, this delay is in addition to the other delays.

When you select the Constant Delay block, the following option is available.

Setting	Description
Delay	The amount of time to delay in seconds.

Dynamic delay block

When trigger model execution reaches a dynamic delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made.

Each measure function can have up to five unique user delay times (M1 to M5). Digitize user delays are handled as measure user delays, so you can have a total of five measure and digitize user delays. The delay time is set by the user-delay command, which is only available over a remote interface. If you are using SCPI, the user delay command is [\[:SENSe\[1\]\]:<function>:DElay:USER<n>](#) (on page 12-139). If you are using TSP, it is [dmm.measure.userdelay\[N\]](#) (on page 14-326).

This delay can be different for every index in a configuration list. This makes it possible to have a delay that changes as a configuration list progresses.

When you select the Dynamic Delay block, the following option is available.

Setting	Description
User Delay	The user delay to recall.

Buffer clear block

When trigger model execution reaches the buffer clear trigger block, the instrument empties the specified reading buffer. The specified buffer can be the default buffer or a buffer that you defined.

For more information about reading buffers, refer to [Reading buffers](#) (on page 6-1).

When you select the buffer clear block, the following option is available.

Setting	Description
Reading Buffer	The name of the buffer to clear.

Config list next block

The config list next block recalls the settings at the next index of a configuration list.

When trigger model execution reaches a configuration recall next block, the settings at the next index in the specified configuration list are restored.

The first time the trigger model encounters this block for a specific configuration list, the first index is recalled. Each subsequent time this block is encountered, the settings at the next index in the configuration list are recalled and take effect before the next step executes. When the last index in the list is reached, it returns to the first index.

The configuration list must be defined before you can use this block.

When you select the Config List Next block, the following option is available.

Setting	Description
Config List	The name of the configuration list from which to recall the next index.

Config list prev block

The Config List Prev block defines a trigger model block that recalls the settings stored at the previous index in a configuration list.

The configuration list previous index trigger block type recalls the previous index in a configuration list. It configures the settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

The first time the trigger model encounters this block, the last index in the configuration list is recalled. Each subsequent time trigger model execution reaches a configuration list previous block for this configuration list, it goes backward one index. When the first index in the list is reached, it goes to the last index in the configuration list.

The configuration list must be defined before you can use this block.

When you select the config list prev block, the following option is available.

Setting	Description
Config List	The name of the configuration list from which to recall the previous index.

Config list recall block

When the trigger model reaches a configuration recall block, the settings in the specified configuration list are recalled.

You can restore a specific set of configuration settings in the configuration list by defining the index.

The configuration list must be defined before you can use this block. If the configuration list changes, verify that the trigger model count is still accurate.

When you select the Config List Recall block, the following options are available.

Setting	Description
Config List	The name of the configuration list to recall the index from.
Recall Index	The index to recall.

Digital input/output block

NOTE

This option requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

The digital I/O block defines a trigger model block that sets the lines on the digital I/O port high or low.

To set the lines on the digital I/O port high or low, you can send an output line bit pattern. The pattern can be specified as an integer value, or, if you are using the TSP command set, a six-bit binary or hexadecimal. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The bit mask defines the bits in the pattern that are driven high or low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63). If the bit for a line in the bit pattern is set to 1, the line is driven high. If the bit is set to 0 in the bit pattern, the line is driven low.

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command). The digital line settings are only available through remote commands.

When you select the digital I/O block, the following options are available.

Setting	Description
Out Line Mask	Sets the output line bit mask (0 to 63).
Out Line Pattern	Sets the value that specifies the output line bit pattern: 0 to 63.

Log event block

This block allows you to log an event in the event log when trigger model execution reaches this block. You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

You can define the type of event (information, warning, abort model, or error). All events generated by this block are logged in the event log. Warning and error events are also displayed in a popup on the front-panel display.

Note that using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger model blocks.

When you select the Log Event block, the following options are available.

Setting	Description
Event Type	The event number or type: ■ Abort Model: Stop the trigger model and log a warning message ■ Information <i>N</i>: Logs an information message in the event log ■ Warning <i>N</i>: Logs a warning in the event log ■ Error <i>N</i>: Logs an error in the event log Where <i>N</i> is 1 to 4; you can define up to four of each type.
Message	A message that you define.

Reset Branch Count

This block creates a block in the trigger model that resets a branch counter to 0.

When you select the Reset Branch Count block, the following option is available.

Setting	Description
Counter Block	Enter a number from 1 to 63.

Notify block

When trigger model execution reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. This assigns a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

NOTE

The TSP-Link and digital I/O options require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Setting up the notify block using the front panel:

When you set up the notify blocks using the front panel, you select the line or timer to notify. You also set specifics regarding the line. The stimulus and logic for input and output lines are set up automatically. The notify event number is also set automatically and is displayed at the bottom of the Notify definition screen.

When the trigger model executes a notify block, the instrument generates the SCPI event `NOTify<n>` or TSP event `trigger.EVENT_NOTIFYN`. You can assign this event to a command that takes an event. For example, if you want a notify block to trigger a digital I/O line, insert a notify block into the trigger model, assign it a notify event and then connect it to the stimulus of the digital I/O line that you want to drive with the notify event.

If you define a LAN trigger from the front panel, you are asked if you want to initiate the LAN connection. You must initiate the connection to use the LAN triggers.

Setting up the notify block using remote commands:

When you set up the notify block using remote commands, you define the notify event number. You need to set up the lines that use the notify event as a stimulus as separate commands.

In the following example, you define trigger model block 5 to be the notify 2 event. You can then assign the notify 2 event to be the stimulus for digital output line 3. To do this, send the following commands in SCPI:

```
:TRIG:BLOC:NOT 5, 2
:TRIG:DIG3:OUT:STIMulus NOTify2
```

In TSP, send the commands:

```
trigger.model.setblock(5, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY2)
trigger.digout[3].stimulus = trigger.EVENT_NOTIFY2
```

If digital I/O line 3 is connected to another instrument, this causes the trigger execution to wait for the other instrument to indicate that it is ready.

Front panel options

When you select the Notify block from the front panel, the following options are available.

Setting	Description
Notify	The line or timer that is notified: ■ Digital Output Line: Select the digital output event (line 1 through line 6) ■ TSP-Link Output Line: Select the TSP-Link output event (line 1 through 3) ■ Timer: Select the timer event (timer 1 through 4) ■ External Trigger Out: Generates a signal through the external trigger out terminal on the DAQ6510 rear panel. ■ LAN Output: Select the LAN output line (line 1 to 8)
Settings	The available settings depend on the Notify selection. For the digital, TSP-Link, external trigger, and LAN output lines, select the pulse logic (negative or positive).

Branching blocks

A branch block goes to a trigger block other than the sequential execution block. For example, if you need to set up the trigger model to wait for an event under some conditions but not others, you can use a branch block to define when the wait block is enabled. You can use the Branch Once block to create a bypass and skip the wait block the first time the trigger model runs. This makes it possible to avoid deadlock when multiple instruments are being synchronized and each one is waiting for notification from the other one to start the trigger model.

Loop Counter block

When trigger model execution reaches a loop counter block, it goes to a specified block until the count value is reached. When the counter exceeds the count value, trigger model execution ignores the branch, continues to the next block in the sequence, and resets the counter.

The counter is reset to 0 when the trigger model starts. It is incremented each time trigger model execution reaches the counter block.

If you are using remote commands, you can query the counter. The counter is incremented immediately before the branch compares the actual counter value to the set counter value. Therefore, the counter is at 0 until the first comparison. When the trigger model reaches the set counter value, branching stops and the counter value is one greater than the setting.

When you select the Loop Counter block, the following options are available.

Setting	Description
Branch to Block	The block number to execute when the counter is less than the Target Count value.
Target Count	The number of times to repeat.

Constant Limit block

The Branch Constant Limit block defines a trigger model block that goes to a specified block if a measurement meets preset criteria.

The measurement block must be a measurement block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from a measurement block is used.

If the limit A is more than the limit B, the instrument automatically swaps the values so that the lesser value is used as the lower limit.

You can use this block to create a binning application by having the block branch to a digital I/O block, followed by a branch always block. Multiple tests can be chained together by repeating this.

NOTE

To use limits that vary programmatically, use the branch-on-dynamic-limits block.

When you select the constant limit block, the following options are available.

Setting	Description
Branch to Block	The block number to execute when the measurement meets the defined criteria.
High Limit	The upper limit that the measurement is compared against. If the type is set to: ■ Inside: The high limit that the measurement is compared against ■ Above: The measurement must be above this value ■ Below: This value is ignored ■ Outside: The high limit that the measurement is compared against
Limit Type	How the limits are compared: ■ Inside: The measurement is inside the values set by limits A and B; limit A must be the low value and Limit B must be the high value ■ Above: The measurement is above the value set by limit B; limit A must be set, but is ignored when this type is selected ■ Below: The measurement is below the value set by limit A; limit B must be set, but is ignored when this type is selected ■ Outside: The measurement is outside the values set by limits A and B; limit A must be the low value and Limit B must be the high value
Low Limit	The lower limit that the measurement is compared against. If the type is set to: ■ Inside: The low limit that the measurement is compared against ■ Above: This value is ignored ■ Below: The measurement must be below this value ■ Outside: The low limit that the measurement is compared against
Measure/Digitize Block	The block number of the block that makes the reading to be compared; from the front panel, you can set this to Previous to use the previous measure/digitize block.

NOTE

When you select a Limit Type of Inside or Outside, two buttons display below the Limit Type button. The button on the left is the Low Limit and the button on the right is the High Limit.

Dynamic Limit block

The branch-on-dynamic-limits block defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

When you define this block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- The limit number (you can have 1 or 2 limits)
- The block to go to if the measurement meets the criteria
- The block that makes the measurement that is compared to the limits; the last measurement from that block is used

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values, which are set using the front-panel Calculations limit settings or through commands. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

Limit values are stored in the measure configuration list, so you can use a configuration list to step through different limit values.

The measure/digitize block must occur in the trigger model before the branch-on-dynamic-limits block. If no block is defined, the measurement from the previous measure/digitize block is used. If no previous measure/digitize block exists, an error is reported.

When you select the dynamic limit block, the following options are available.

Setting	Description
Branch to Block	The block number to execute when the measurement meets the defined criteria.
Limit Number	The limit that is used for this block, 1 or 2.
Limit Type	How the limits are compared: ■ Inside: The measurement is within the limits ■ Above: The measurement is above the high limit ■ Below: The measurement is below the low limit ■ Outside: The measurement is outside the limits
Measure/Digitize Block	The block number of the block that makes the reading to be compared; from the front panel, you can set this to Previous to use the previous measure/digitize block.

Once block

When the trigger model reaches a branch-once block, it goes to a specified block the first time it is encountered in the trigger model. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

You can use this block to create a bypass. For example, you might place a branch-once block before a wait block to skip the wait block on the first pass of the trigger model.

The once block is reset when the trigger model reaches the idle state. Therefore, the branch-once block will always execute the first time the trigger model encounters this block.

When you select the Once block, the following option is available.

Setting	Description
Branch to Block	The block number to execute the first time the trigger model reaches the Once block.

Once excluded block

The branch-once-excluded block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts or is placed in idle.

When you select the once excluded block, the following option is available.

Setting	Description
Branch to Block	The block number to execute after the first time the trigger model reaches the once excluded block.

Delta block

The branch on delta block defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

This block calculates the difference between the last two measurements from a measure/digitize block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measure/digitize block, it will compare measurements of a measure/digitize block that precedes the branch delta block. For example, if you have a measure/digitize block, a wait block, another measure/digitize block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure/digitize block. If a preceding measure/digitize block does not exist, an error occurs.

When you select the delta block, the following options are available.

Setting	Description
Branch to Block	The block number of the trigger model block to execute when the difference between the measurements is less than or equal to the Target Delta.
Measure/Digitize Block	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block.
Target Delta	The value against which the block compares the difference between the measurements.

On event block

The branch-on-event block branches to a specified block when a specified trigger event occurs. If the trigger event has not yet occurred when trigger model execution reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time trigger model execution reaches the branch-on-event block, it goes to the branching block.

Trigger events are reset when the trigger model is at the start block, so only events that occur after you start trigger model execution are detected by the branch-on-event block. The event is also reset after trigger model execution completes the branching block.

When you select the branch-on-event block, the following options are available.

Setting	Description
Branch to Block	The block number to execute when the specified event occurs.
Event	The event that causes this block to branch.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

The event can be any of the events described in the following table.

Event	Description
Analog Trigger	Use the analog trigger.
Blender	Wait for the events set by an event blender.
Command	A command interface trigger (bus trigger): ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command <code>device_trigger</code>
Digital Input	Line edge detected on a digital input line. When you select this option, you select the digital input to monitor. After selecting the digital input line, choose Settings to select the type of edge (falling, rising, or either).
Display TRIGGER Key	Front-panel TRIGGER key press.
External Trigger In	Use a pulse from the external trigger. When you select this option, choose Settings to select the type of edge (falling, rising, or either).
LAN Input Trigger	An LXI trigger packet is received on the LAN trigger object. When you select this option, you select the LAN trigger to monitor. After you select the line, choose Settings to select the type of edge (falling, rising, or either).
None	No trigger event.
Timer	A trigger timer expired. When you select this option, you select the timer to monitor. After selecting the timer, choose Settings to define the timer. Refer to Trigger timers (on page 8-26) for detail on the options. For a timer to expire, you must start it. One method to start the timer in the Trigger Model is to include a Notify block before this block. Set the Notify block to use the same timer.
TSP-Link Input	Line edge detected on a TSP-Link synchronization line. When you select this option, you select the TSP-Link line to monitor. After selecting the TSP-Link line, choose Settings to select the type of edge (falling, rising, or either).

For information on trigger events, see [Using trigger events to start actions in the trigger model](#) (on page 8-69).

Always block

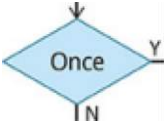
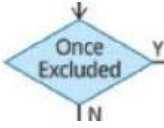
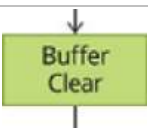
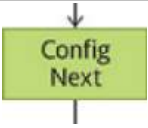
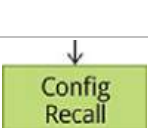
When the trigger model reaches a branch-always block, it goes to the block that you specified.

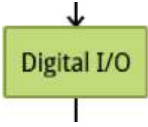
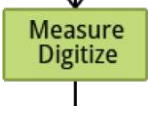
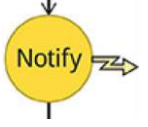
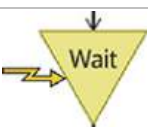
When you select the always block, the following option is available.

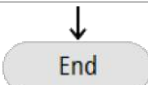
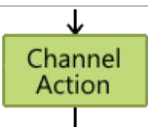
Setting	Description
Branch to Block	The block number to execute when the trigger model reaches the always block.

Trigger block summary

Front-panel icon	SCPI command TSP command	Block description
Not applicable	:TRIGger:BLOCK:LIST? (on page 12-294) trigger.model.getblocklist() (on page 14-476)	This returns the settings for all trigger model blocks
	:TRIGger:BLOCK:BRANch:ALWayS (on page 12-274) trigger.model.setblock() — trigger.BLOCK_BRANCH_ALWAYS (on page 14-493)	This defines a trigger model block that always goes to a specific block
	:TRIGger:BLOCK:BRANch:COUNter (on page 12-274) trigger.model.setblock() — trigger.BLOCK_BRANCH_COUNTER (on page 14-493)	This defines a trigger model block that branches to a specified block a specified number of times
Not applicable	:TRIGger:BLOCK:BRANch:COUNter:COUNt? (on page 12-275) trigger.model.getbranchcount() (on page 14-477)	This returns the count value of the trigger model counter block
	:TRIGger:BLOCK:BRANch:COUNter:RESEt (on page 12-276) trigger.model.setblock() — trigger.BLOCK_RESET_BRANCH_COUNT (on page 14-517)	This creates a block in the trigger model that resets a branch counter to 0
	:TRIGger:BLOCK:BRANch:DELTA (on page 12-277) trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA (on page 14-494)	This defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria
	:TRIGger:BLOCK:BRANch:EVENT trigger.model.setblock() — trigger.BLOCK_BRANCH_ON_EVENT (on page 14-500)	This branches to a specified block when a specified trigger event occurs
	:TRIGger:BLOCK:BRANch:LIMit:CONStant (on page 12-281) trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_CONSTANT (on page 14-496)	This defines a trigger model block that goes to a specified block if a measurement meets preset criteria
	:TRIGger:BLOCK:BRANch:LIMit:DYNamic (on page 12-283) trigger.model.setblock() — trigger.BLOCK_BRANCH_LIMIT_DYNAMIC (on page 14-498)	This defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria

Front-panel icon	SCPI command TSP command	Block description
	:TRIGger:BLOCK:BRANCh:ONCE (on page 12-285) trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE (on page 14-502)	This causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model
	:TRIGger:BLOCK:BRANCh:ONCE:EXCLUded (on page 12-286) trigger.model.setblock() — trigger.BLOCK_BRANCH_ONCE_EXCLUDED (on page 14-502)	This causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time
	:TRIGger:BLOCK:BUFFer:CLEar (on page 12-287) trigger.model.setblock() — trigger.BLOCK_BUFFER_CLEAR (on page 14-503)	This defines a trigger model block that clears the reading buffer
	:TRIGger:BLOCK:CONFIg:NEXT (on page 12-288) trigger.model.setblock() — trigger.BLOCK_CONFIG_NEXT (on page 14-504)	This recalls the settings at the next index of a configuration list
	:TRIGger:BLOCK:CONFIg:PREVious (on page 12-289) trigger.model.setblock() — trigger.BLOCK_CONFIG_PREV (on page 14-505)	This defines a trigger model block that recalls the settings stored at the previous index in a configuration list
	:TRIGger:BLOCK:CONFIg:RECall (on page 12-290) trigger.model.setblock() — trigger.BLOCK_CONFIG_RECALL (on page 14-506)	This recalls the system settings that are stored in a configuration list
	:TRIGger:BLOCK:DELAy:CONStant (on page 12-291) trigger.model.setblock() — trigger.BLOCK_DELAY_CONSTANT (on page 14-507)	This adds a constant delay to the execution of a trigger model
	:TRIGger:BLOCK:DELAy:DYNamic (on page 12-292) trigger.model.setblock() — trigger.BLOCK_DELAY_DYNAMIC (on page 14-508)	This adds a user delay to the execution of the trigger model

Front-panel icon	SCPI command TSP command	Block description
	:TRIGger:BLOCK:DIGital:IO (on page 12-293) trigger.model.setblock() — trigger.BLOCK_DIGITAL_IO (on page 14-509)	This trigger model block that sets the lines on the digital I/O port high or low
	:TRIGger:BLOCK:LOG:EVENT (on page 12-294) trigger.model.setblock() — trigger.BLOCK_LOG_EVENT (on page 14-510)	This allows you to log an event in the event log when the trigger model is running
	:TRIGger:BLOCK:MDIGitize (on page 12-295) trigger.model.setblock() — trigger.BLOCK_MEASURE_DIGITIZE (on page 14-511)	This defines a trigger block that makes or digitizes a measurement, depending on the active function
	:TRIGger:BLOCK:NOP (on page 12-298) trigger.model.setblock() — trigger.BLOCK_NOP (on page 14-515)	This creates a placeholder that performs no action in the trigger model; available only using remote commands
	:TRIGger:BLOCK:NOTify (on page 12-298) trigger.model.setblock() — trigger.BLOCK_NOTIFY (on page 14-516)	This defines a trigger model block that generates a trigger event and immediately continues to the next block
	:TRIGger:BLOCK:WAIT (on page 12-300) trigger.model.setblock() — trigger.BLOCK_WAIT (on page 14-518)	This defines a trigger model block that waits for an event before allowing the trigger model to continue

Front-panel icon	SCPI command TSP command	Block description
	Not applicable	This block is set up internally as part of the scan and cannot be changed using commands or the front panel.
	Not applicable	This block is set up internally as part of the scan and cannot be changed using commands or the front panel.
	Not applicable	This block is set up internally as part of the scan and cannot be changed using commands or the front panel.

Trigger-model templates

The DAQ6510 includes trigger-model templates for common applications. You can use these templates without changing them, or you can modify them to meet the needs of your application.

The trigger-model templates include:

- **Empty:** Clears the present trigger model.
- **ConfigList:** Creates a trigger model that loads a configuration list. At each configuration list index, a measurement is made. The list is iterated until every index in the configuration list has been loaded.
- **LogicTrigger:** Creates a trigger model that waits on an input line, delays, makes a measurement, and sends out a trigger on the output line a specified number of times.
- **SimpleLoop:** Creates a trigger model that makes a specified number of readings. A count parameter defines the number of readings.
- **DurationLoop:** Creates a trigger model that makes continuous measurements for a specified amount of time.
- **LoopUntilEvent:** Creates a trigger model that makes continuous measurements until a specified event occurs.
- **GradeBinning:** Creates a trigger model that successively measures components and compares their readings to high or low limits to grade components. Only useable if a communications accessory card is installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.
- **SortBinning:** Creates a trigger model that successively measures components and compares their readings to high or low limits to sort components. Only useable if a communications accessory card is installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

NOTE

Settings for the GradeBinning and SortBinning templates are shared. If you switch between these two templates, be aware that settings are retained.

Preparations for using a trigger-model template

Before starting the trigger model, you need to set up your instrument for testing, including the measure or digitize settings and configuration lists.

When you load a trigger-model template, the instrument overwrites any existing trigger models.

NOTE

If you select Templates and only the Empty trigger model appears, select Empty to see the full list of template options.

Using a trigger-model template to develop a trigger model

The DAQ6510 includes trigger-model templates that you can use as a starting point for developing your trigger model.

After modifying a trigger-model template, you can save it in a saved setup for future use. See [Saving setups](#) (on page 4-92) for information on how to save a configuration.

The trigger model is changed to the selected template when you change a setting, press the HOME key, or press the MENU key.

If you select a trigger-model template, but then customize it using options on the Trigger Configure menu, Templates displays *Custom*.

Using the front panel:

1. Press the **MENU** key.
2. Under Trigger, select **Templates**. The TRIGGER MODEL TEMPLATES screen is displayed.
3. Use **Templates** to select the trigger-model template.
4. Change the settings for the template as needed.
5. Select **MENU**.
6. Under Trigger, choose **Configure**. The blocks for the trigger-model template are displayed.
7. Modify the blocks as needed. See [Assembling trigger-model blocks](#) (on page 8-62).
8. When the blocks are set up, select **HOME**.
9. Select the measurement method indicator and choose **Manual Trigger Model**. This sets the TRIGGER key to initiate the trigger model. See [Measurement method indicator](#) (on page 3-14) for additional information.
10. Press the **TRIGGER** key to initiate the trigger model.

Using SCPI commands:

See the descriptions of the `TRIGger:LOAD` commands for details on the options available for each trigger-model template.

- [:TRIGger:LOAD "ConfigList"](#) (on page 12-322)
- [:TRIGger:LOAD "DurationLoop"](#) (on page 12-323)
- [:TRIGger:LOAD "Empty"](#) (on page 12-324)
- [:TRIGger:LOAD "GradeBinning"](#) (on page 12-325)
- [:TRIGger:LOAD "LogicTrigger"](#) (on page 12-327)

- [:TRIGger:LOAD "LoopUntilEvent"](#) (on page 12-328)
- [:TRIGger:LOAD "SimpleLoop"](#) (on page 12-330)
- [:TRIGger:LOAD "SortBinning"](#) (on page 12-331)

Using TSP commands:

See the descriptions of the `trigger.model.load()` command for details on the options available for each trigger-model template.

- [trigger.model.load\(\) — ConfigList](#) (on page 14-478)
- [trigger.model.load\(\) — DurationLoop](#) (on page 14-480)
- [trigger.model.load\(\) — Empty](#) (on page 14-481)
- [trigger.model.load\(\) — GradeBinning](#) (on page 14-481)
- [trigger.model.load\(\) — LogicTrigger](#) (on page 14-484)
- [trigger.model.load\(\) — LoopUntilEvent](#) (on page 14-485)
- [trigger.model.load\(\) — SimpleLoop](#) (on page 14-487)
- [trigger.model.load\(\) — SortBinning](#) (on page 14-488)

Assembling trigger-model blocks

This section describes the basic concepts you need to understand to assemble trigger-model blocks.

Sequencing trigger-model blocks

You can set up trigger-model blocks from the front panel or by using remote commands.

Trigger-model blocks must be sequenced in order; you cannot skip numbers. When the trigger model completes the last block in the trigger model, the trigger model returns to the idle state. Idle is considered to be execution block 0. Branching to block 0 effectively stops the trigger model.

As the trigger model reaches each block, the action defined by that block is started and completed before the trigger model moves to the next block. Blocks do not overlap.

The trigger model steps through the blocks in sequential order. You can set up branching blocks to allow nonsequential actions to occur. See [Branching blocks](#) (on page 8-47) for detail on how to use the branching blocks.

If you skip block numbers, when you initiate the trigger model, the trigger model generates an event message that reports the missing block. You can view and delete the missing blocks on the front-panel TriggerFlow® screen. If you delete them using the front-panel options, the remaining blocks are resequenced.

You can have up to 63 blocks in a trigger model.

Working with the trigger model

You can change existing trigger-model blocks through the front panel or by sending a remote command. The blocks are redefined with the new parameters.

When you define the trigger model using remote commands, you can send blocks in any order. For example, you can define block 5 before defining blocks 1 to 4. However, you cannot run a trigger model with undefined blocks.

If you skipped a block, you can use the no operation block to define a block that will not affect the trigger model and save the effort of resequencing the other blocks. The no operation block is available through the remote commands only (SCPI command [:TRIGger:BLOCK:NOP](#) (on page 12-298) or TSP command [trigger.model.setblock\(\) — trigger.BLOCK NOP](#) (on page 14-515)).

Determining the structure of the existing trigger model

You can retrieve the existing trigger model structure from the front panel or by using remote commands.

Using the front panel:

1. Press the **MENU** key.
2. Under Trigger, select **Configure**. The trigger model is displayed.
3. If the trigger model is longer than one screen, swipe the TriggerFlow diagram to scroll up or down.
4. To view the settings for a block, select the block. The settings are displayed on the right.
5. For a description of a setting, press and hold the button and press the **HELP** key.

For additional information on the blocks, refer to the block descriptions under [Trigger model blocks](#) (on page 8-37).

Using SCPI commands:

To retrieve the settings for all trigger model blocks, send the command:

```
:TRIGger:BLOCK:LIST?
```

Using TSP commands:

To check the settings for a block, send the command:

```
print(trigger.model.getblocklist())
```

NOTE

To retrieve the TSP code for trigger model blocks that are entered through the front panel, change the Event Log "Command" setting to On. Refer to [Using the event log](#) (on page 3-80) for additional information.

Improving the performance of a trigger model

To improve the performance of a trigger model:

- Reduce the number of blocks to less than 15.
- Do not use multiple reading buffers.
- Use four or fewer delay blocks.
- Use four or fewer measure/digitize blocks.
- Do not have multiple blocks waiting on the same event.
- Verify that constant delay blocks are set to less than 254 ms.
- Limit use of configuration list blocks.

Action overruns

An action overrun occurs when a trigger object receives a trigger event and is not ready to act on it. The action overruns of all trigger objects are reported in a command for the associated trigger object. See the appropriate sections on each trigger object for further details on conditions under which an object generates an action overrun.

Some examples of action overruns include the following:

- `trigger.blender[N].overrun`
- `trigger.digin[N].overrun`
- `trigger.extin.overrun`
- `trigger.lanin[N].overrun`
- `trigger.timer[N].overrun`
- `trigger.tsplinkin[N].overrun`

Running the trigger model

You can run the trigger model from the front panel or by using remote commands.

When you run the trigger model, the existing instrument settings are used for any actions unless you assigned configuration lists to the trigger model.

Trigger-model operation is an overlapped process. This means that you can run other commands while a trigger model is running if they do not conflict with trigger-model operation. For example, you can print the buffer contents, but you cannot change the measure function.

The initiate command is the overlapped command that starts the process. The command interface is available immediately after the instrument executes the initiate command so that other commands can be executed while the trigger model is running.

If you change from remote to local control, the trigger model measurement method remains selected until you change it. To change the measurement method, see [Switching between measurement methods](#) (on page 4-47).

If you change from remote to local control or from local to remote control while a trigger model is running, the trigger model is aborted.

Starting the trigger model

Using the front panel:

1. Press the front-panel **TRIGGER** key for 2 s. A dialog box displays the available trigger methods. The presently selected method is highlighted.
2. Select **Initiate Trigger Model**.
3. If the instrument is controlled remotely, a confirmation screen is displayed. Select **Yes** to change to front-panel control and start the trigger model.

Using SCPI commands:

Send the command:

```
:INITiate
```

Using TSP commands:

Send the command:

```
trigger.model.initiate()
```

Aborting the trigger model

You can stop the trigger model while it is in progress. When you stop the trigger model, all trigger model commands on the instrument are terminated.

Using the front panel:

Press the **TRIGGER** key for two seconds and select **Abort Trigger Model**.

Using SCPI commands:

Send the command:

```
:ABORt
```

Using TSP commands:

Send the command:

```
trigger.model.abort()
```

Pausing and resuming the trigger model

You can pause the trigger model or scan while it is in progress by using the pause command. To restart the trigger model and the scan after pausing, use the resume command.

Using the front panel:

On the SCAN swipe screen, select **Pause Scan**.

To start the scan again, select **Resume Scan**.

Using SCPI commands:

To pause, send the command:

```
:TRIGger:PAUSe
```

To restart, send the command:

```
:TRIGger:RESume
```

Using TSP commands:

To pause, send the command:

```
trigger.model.pause()
```

To restart, send the command:

```
trigger.model.resume()
```

Checking the state of the trigger model

The trigger model can be in one of several states. The state is shown in the indicator bar on the home screen of the instrument. You can also check the status using remote commands.

The following table describes the trigger model states. This table also describes the indicator that is shown on the front panel and the feedback you get from the remote interface.

Front-panel indicator	SCPI remote command feedback	TSP remote command feedback	Description
N/A	ABORTED	trigger.STATE_ABORTED	The trigger model was stopped before it completed
CONT	Not available through remote interface	Not available through remote interface	Instrument is not using the trigger model; it is making measurements continuously
IDLE	IDLE or EMPTY	trigger.STATE_IDLE or trigger.STATE_EMPTY	Trigger model is stopped, or no blocks are defined
INACT	INACTIVE	trigger.STATE_INACTIVE	Instrument encountered system settings that do not yield a reading
MAN	Not available through remote interface	Not available through remote interface	Instrument is not using trigger model; makes measurements when you press the front-panel TRIGGER key
PAUSE	PAUSE	trigger.STATE_PAUSE	Trigger model is paused
RUN	RUNNING	trigger.STATE_RUNNING	Trigger model is running
WAIT	WAITING	trigger.STATE_WAITING	The trigger model has been in the wait block for more than 100 ms

Using the front panel

The state of the trigger model is indicated on the status bar with the indicators shown in the previous table.

Using SCPI commands:

Send the command:

```
:TRIGger:STATe?
```

The return shows the state and the block that was last executed.

Using TSP commands:

Send the command:

```
print(trigger.model.state())
```

The return shows the state and the block that was last executed.

Using trigger events to start actions in the trigger model

You can set up trigger blocks to respond to trigger events. Trigger events are signals that can be generated by the instrument or by other system components.

Sources of the trigger event signals can be:

- Front-panel TRIGGER key
- Notify trigger blocks
- Branch-on-event trigger blocks
- Command-interface triggers
- Digital I/O lines (requires a communications card)
- TSP-Link synchronization lines (requires the KTTI-TSP communications card)
- LAN triggers
- Analog triggers
- EXTERNAL TRIGGER IN or EXTERNAL TRIGGER OUT signals
- Event blenders, which combine other trigger events
- Trigger timers

For information about the options that are not specific to the trigger model, see [Triggering](#) (on page 8-18).

Trigger events

To use trigger events, you need to specify the event constant. The tables below show the constants for the trigger events in the system. You can use these events with instrument features such as trigger timers, trigger blocks, digital I/O lines, and external I/O lines.

Trigger events: SCPI command set

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMManD
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENDer<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPLete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmLimit (returns NOT3)

Trigger events: TSP command set

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	trigger.EVENT_NONE
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	trigger.EVENT_COMMAND
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	trigger.EVENT_DIGION
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	trigger.EVENT_TSPLINKN
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	trigger.EVENT_LANN

Trigger events	
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender N (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer N (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Scan alarm limit exceeded	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Channel closed	<code>trigger.EVENT_SCAN_CHANNEL_READY</code> (returns <code>trigger.EVENT_NOTIFY6</code>)
Scan completed	<code>trigger.EVENT_SCAN_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY8</code>)
Measure completed	<code>trigger.EVENT_SCAN_MEASURE_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY7</code>)
Limit value for scan reached	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code> (returns <code>trigger.EVENT_NOTIFY3</code>)

Using the TRIGGER key to generate an event

You can use the front-panel TRIGGER key to generate a trigger event.

For example, if you set a wait block to advance when the TRIGGER key is pressed, the trigger model will reach the wait block. If the TRIGGER key has already been pressed, the trigger model execution will continue. If the TRIGGER key has not been pressed, the trigger model execution is halted until the TRIGGER key is pressed.

To set a trigger block to respond to the front-panel key press:

- **From the front panel:** Set the event to be Display TRIGGER Key
- **Using SCPI:** Set the event to `DISPlay`
- **Using TSP:** Set the event to `trigger.EVENT_DISPLAY`

There are no action overruns for front-panel TRIGGER key events.

Respond to an event with a wait block

The wait building block causes the trigger model to stop and wait for an event or set of events to occur before continuing. You can specify up to three events for each wait block. The wait block can use any of the system trigger events. See [Trigger events](#) (on page 8-69).

To continue the trigger model, it must receive the trigger event that is defined for the wait block.

Using the branch-on-event trigger blocks

The branch-on-event block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

If you set the branch event to none, an error is generated when you run the trigger model.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a Notify block before the On Event block. Set the Notify block to use the same timer as the On Event block.

The branch-on-event block can use any of the system trigger events. See [Trigger events](#) (on page 8-69).

TSP-Link and TSP-Net

In this section:

TSP-Link System Expansion Interface	9-1
TSP-Net	9-18

TSP-Link System Expansion Interface

Keithley TSP-Link® is a high-speed trigger synchronization and communications bus that test system builders can use to connect multiple instruments in a master and subordinate configuration. Once connected, all the instruments that are equipped with TSP-Link in a system can be programmed and operated under the control of the master instrument or instruments. This allows the instruments to run tests more quickly because they can be decoupled from frequent computer interaction. The test system can have multiple master and subordinate groups, which can be used to handle multi-device testing in parallel. Combining TSP-Link with a flexible programmable trigger model ensures speed.

Using TSP-Link, multiple instruments are connected and can be used as if they are part of the same physical unit for simultaneous multichannel testing. The test system can be expanded to include up to 32 TSP-Link enabled instruments.

TSP-Link functionality is only available when using the instrument front panel or the TSP commands to control the instrument. It is not available if you are using SCPI commands.

TSP-Link connections

The DAQ6510 has three synchronization lines that are built into the TSP-Link connection. If you are using a TSP-Link network, you do not have to modify any connections.

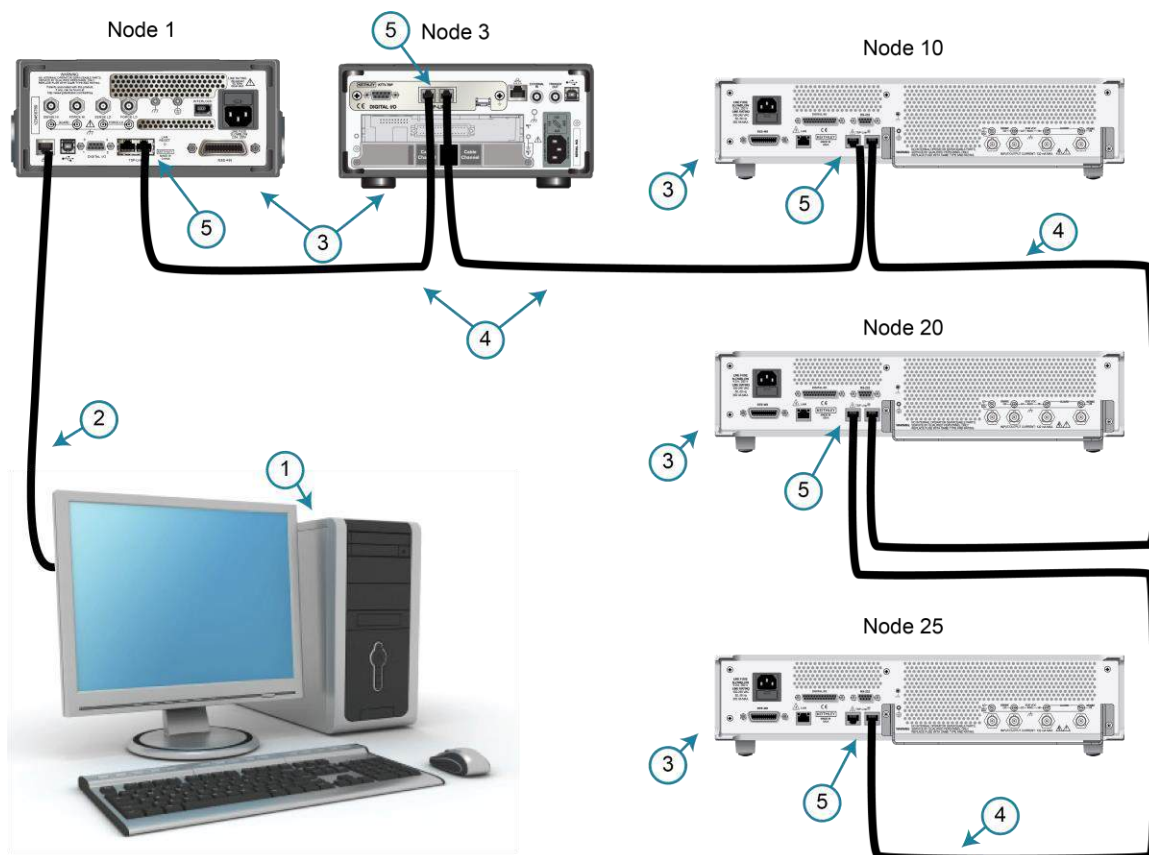
Example connections for a TSP-Link system are shown in the following figure.

The TSP-Link connectors are on the rear panel of the instruments. All the instruments in the system are connected in a sequence (daisy-chained) using LAN crossover cables.

NOTE

If you have a USB communications cable connected to the instrument, you must disconnect it before using TSP-Link connections.

Figure 149: TSP-Link connections



Item	Description	Notes
1	Controller	Optional. A computer is not needed for stand-alone systems.
2	Communications connection	Optional. Connection from controller to the master node through GPIB, LAN, or USB. Details about these computer communications connections are described in Remote communications interfaces (on page 2-7).
3	Nodes	You can have up to 32 nodes on the TSP-Link system. Each node must have a unique node number from 1 to 63.
4	LAN crossover cable	Type 5e category or higher; 3 meters (9.8 feet) maximum between nodes.
5	TSP-Link connections	Each instrument has two TSP-Link connectors. You can make the connection to either TSP-Link connection.

TSP-Link nodes

Each instrument or enclosure attached to the TSP-Link expansion interface is called a node. Each node must be identified with a unique node number. This identification is called a TSP-Link node number.

An individual node is accessed as `node[N]`, where *N* is the node number assigned to the node. You can access all TSP-accessible remote commands as elements of the specific node. The following attributes are examples of items you can access:

- `node[N].model`: The product model number of the node.
- `node[N].version`: The product version of the node.
- `node[N].serialno`: The product serial number of the node.

Assigning node numbers

Each DAQ6510 instrument is initially assigned as node 2. You can assign node numbers from 1 to 63. However, the system can only include 32 physical nodes.

The node number for each instrument is stored in its nonvolatile memory and remains in storage when the instrument is turned off.

You can assign a node number to an instrument using the front panel or by using a remote command.

To assign a node number using the front panel:

1. Press the **MENU** key.
2. Under System, select **Communication**. The SYSTEM COMMUNICATIONS window opens.
3. Select the **TSP-Link** tab.
4. Next to Node, set the TSP-Link address for this instrument.

To assign a node number using a remote command:

Set the `tsplink.node` attribute of the instrument:

```
tsplink.node = N
```

Where $N = 1$ to 63

To determine the node number of an instrument, you can read the `tsplink.node` attribute by sending the following command:

```
print(tsplink.node)
```

The above `print` command outputs the node number. For example, if the node number is 1, a 1 is displayed.

Master and subordinates

In a TSP-Link system, one of the nodes (instruments) is the master node and the other nodes are the subordinate nodes. The master node in a TSP-Link system can control the other nodes (subordinates) in the system.

A TSP-Link system can be stand-alone or computer-based.

In a stand-alone system, scripts are loaded into the instruments. You can run a script from the front panel of any instrument (node) connected to the system. When a script is run, all nodes in the system go into remote operation. When the script is finished running, all the nodes in the system return to local operation, and the master/subordinate relationship between nodes is dissolved.

In a computer-based system, you can use a computer and a remote interface to communicate with a single node in the system. This node becomes the interface to the entire system. When a command is sent through this node, all nodes go into remote operation. The node that receives the command becomes the master and can control all other nodes, which become its subordinates. In a computer-based system, the master/subordinate relationship between nodes can only be dissolved by performing an abort operation. For more information about remote interfaces, see [Remote communications interfaces](#) (on page 2-7).

NOTE

When linking with earlier models of Keithley, such as the Model 2600B, make sure to use the DAQ6510 as the master node and the earlier instruments as subordinates.

Initializing the TSP-Link system

The TSP-Link system must be initialized after configuration changes. You need to initialize the system after you:

- Turn off power or reboot any instrument in the system
- Change node numbers on any instrument in the system
- Rearrange or disconnect the TSP-Link cable connections between instruments

If initialization is not successful, you can check the event log for error messages that indicate the problem. Some typical problems include:

- Two or more instruments in the system have the same node number
- There are no other instruments connected to the instrument performing the initialization
- One or more of the instruments in the system is turned off
- The actual number of nodes is less than the expected number

From the front panel:

1. Power on all instruments connected to the TSP-Link network.
2. Press the **MENU** key.
3. Under System, select **Communication**. The SYSTEM COMMUNICATIONS window opens.
4. Select the **TSP-Link** tab.
5. Select **Initialize**.

Using TSP commands:

To initialize the TSP-Link system, send the command:

```
tsplink.initialize()
```

To check the state of the TSP-Link system, send the command:

```
print(tsplink.state)
```

If initialization was successful, `online` is returned. If initialization was not successful, `offline` is returned.

Sending commands to TSP-Link nodes

You can send remote commands to any instrument on the TSP-Link system by adding `node[N].` to the beginning of the remote command, where *N* is the node number.

For example, to sound the beeper on node 10, you would send the command:

```
node[10].beeper.beep(2, 2400)
```

To send a command to the master, you can interact with it as if it were a single instrument.

Using the `reset()` command

Most TSP-Link® system operations target a single node in the system, but the `reset()` command affects the system as a whole by resetting all nodes to their default settings:

```
-- Reset all nodes in a TSP-Link system to their default state.
reset()
```

NOTE

Using the `reset()` command in a TSP-Link network differs from using the [tsplink.initialize\(\)](#) (on page 14-543) command. The `tsplink.initialize()` command reinitializes the TSP-Link network and turns off the output of any TSP-linked instrument. It may change the state of individual nodes in the system.

Use `node[N].reset()` or `localnode.reset()` to reset only one of the nodes. The other nodes are not affected. The following programming example shows this type of reset operation with code that is run on node 1.

```
-- Reset node 1 only.
node[1].reset()
-- Reset the node you are connected to (in this case, node 1).
localnode.reset()
-- Reset node 4 only.
node[4].reset()
```

Terminating scripts on the TSP-Link system

You can terminate a script that is executing on a TSP-Link system.

To terminate an executing script and return all nodes to local control, send the following command:

```
abort
```

This dissolves the master/subordinate relationships between nodes.

From the front panel, you can abort a script by pressing the TRIGGER key for a few seconds and selecting **Abort Trigger Model** from the dialog box that is displayed.

Triggering using TSP-Link trigger lines

The DAQ6510 has three trigger lines that you can use for triggering, digital I/O, and to synchronize multiple instruments on a TSP-Link® network.

Using TSP-Link trigger lines for digital I/O

Each trigger line is an open-drain signal. When using the TSP-Link® trigger lines for digital I/O, any node that sets the programmed line state to zero (0) causes all nodes to read 0 from the line state. This occurs regardless of the programmed line state of any other node. Refer to the table in [Digital I/O bit weighting](#) (on page 8-13) for digital bit-weight values.

Transferring data

You cannot transfer data to the master in real time at high speeds. Bus errors will occur if these types of data transfers occur.

To avoid the bus errors, transfer data after the measurement acquisition completes. To avoid accidentally configuring a real time transfer, specify the node prefix when entering the buffer to store measurements. A subordinate should store high speed digitizer data in `node[N].buffername` instead of `buffername` when executing scripts on the master. If the `node[N]` prefix is missing, it attempts to store data in the buffer of the master node, which requires measurements to be transferred at acquisition speeds and overruns the bus.

Running simultaneous test scripts

Running test scripts simultaneously improves functional testing, provides higher throughput, and expands system flexibility. You can use TSP-Link and TSP scripting to run simultaneous test scripts. You can also manage the resources that are allocated to test scripts that are running simultaneously.

In addition, you can use the data queue to do real-time communications between nodes on the TSP-Link system.

To run test scripts simultaneously, you can set up your TSP-Link network in one of the following configurations:

- Multiple TSP-Link networks
- Single TSP-Link network with groups

Using groups to manage nodes on a TSP-Link system

TSP-Link groups allow each group to run a different test script simultaneously. This method requires one TSP-Link network and a single remote connection to the computer that is connected to the master node.

A group can consist of one or more nodes. You must assign group numbers to each node using remote commands. If you do not assign a node to a group, it defaults to group 0, which will always be grouped with the master node (regardless of the group to which the master node is assigned).

The following table shows an example of the functions of groups on a single TSP-Link network. Each group in this example runs a different test script than the other groups, which allows the system to run multiple tests simultaneously.

TSP-Link network group functions

Group number	Group members	Present function
0	Master node 1	Initiates and runs a test script on node 2 Initiates and runs a test script on node 6 In addition, the master node can execute scripts and process run commands
1	Group leader node 2	Runs the test script initiated by the master node Initiates remote operations on node 3 through node 5
	Node 3 through node 5	Performs remote operations initiated by node 2
2	Group leader node 6	Runs the test script initiated by the master node Initiates remote operations on node 7 through node <i>n</i>
	Node 7 through node <i>n</i>	Performs remote operations initiated by node 6

Master node overview

You can assign the master node to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to group 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

The master node is always the node that coordinates activity on the TSP-Link network.

The master node:

- Is the only node that can use the `execute()` command on a remote node
- Cannot initiate remote operations on any node in a remote group if any node in that remote group is performing an overlapped operation (a command that continues to operate after the command that initiated it has finished running)
- Can execute the `waitcomplete()` command to wait for the group to which the master node belongs; to wait for another group; or to wait for all nodes on the TSP-Link network to complete overlapped operations (overlapped commands allow the execution of subsequent commands while device operations of the overlapped command are still in progress)

Group leader overview

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

The group leader:

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

Assigning groups

Group numbers can range from zero (0) to 64. The default group number is 0. You can change the group number at any time. You can also add or remove a node to or from a group at any time.

Each time the power for a node is turned off, the group number for that node changes to 0.

The following example code dynamically assigns a node to a group:

```
-- Assign node 3 to group 1.  
node[3].tsplink.group = 1
```

Running test scripts and programs on remote nodes

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

Use the following code to send the `execute()` command for a remote node. The *N* parameter represents the node number that runs the test script (replace *N* with the node number).

To set the global variable "setpoint" on node *N* to 2.5:

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

The following code demonstrates how to run a test script that is defined on the local node. For this example, *scriptVar* is defined on the local node, which is the node that initiates the code to run on the remote node. The local node must be the master node.

To run *scriptVar* on node *N*:

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

The programming example below demonstrates how to run a test script that is defined on a remote node. For this example, *scriptVar* is defined on the remote node.

To run a script defined on the remote node:

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

It is recommended that you copy large scripts to a remote node to improve system performance.

Coordinating overlapped operations in remote groups

All overlapped operations on all nodes in a group must have completed before the master node can send a command to the group. If you send a command to a node in a remote group when an overlapped operation is running on any node in that group, errors occur.

You can execute the `waitcomplete()` command on the master node or group leader to wait for overlapped operations. The action of `waitcomplete()` depends on the parameters specified.

If you want to wait for completion of overlapped operations for:

- **All nodes in the local group:** Use `waitcomplete()` without a parameter from the master node or group leader.
- **A specific group:** Use `waitcomplete(N)` with a group number as the parameter from the master node. This option is not available for group leaders.
- **All nodes in the system:** Use `waitcomplete(0)` from the master node. This option is not available for group leaders.

For additional information, refer to [waitcomplete\(\)](#) (on page 14-572).

The following code shows two examples of using the `waitcomplete()` command from the master node:

```
-- Wait for each node in group N to complete all overlapped operations.
waitcomplete(N)
-- Wait for all groups on the TSP-Link network to complete overlapped operations.
waitcomplete(0)
```

A group leader can issue the `waitcomplete()` command to wait for the local group to complete all overlapped operations.

The following code is an example of how to use the `waitcomplete()` command from a group leader:

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


Using the data queue for real-time communications

Nodes that are running test scripts at the same time can store data in the data queue for real-time communications. Each instrument has an internal data queue that uses the first-in, first-out (FIFO) structure to store data. You can use the data queue to post numeric values, strings, and tables.

Use the data queue commands to:

- Share data between test scripts running in parallel
- Access data from a remote group or a local node on a TSP-Link network at any time

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

Tables in the data queue consume one entry. When a node stores a table in the data queue, a copy of the data in the table is made. When the data is retrieved from the data queue, a new table is created on the node that is retrieving the data. The new table contains a separate copy of the data in the original table, with no references to the original table or any subtables.

You can access data from the data queue even if a remote group or a node has overlapped operations in process. See the `dataqueue` commands for more information.

Remote TSP-Link commands

Commands that control and access the TSP-Link® synchronization port are summarized in the following table. See the [TSP command reference](#) (on page 14-1) for complete details on these commands.

Use the commands in the following table to perform basic steady-state digital I/O operations; for example, you can program the DAQ6510 to read and write to a specific TSP-Link synchronization line or to the entire port.

TSP-Link commands

Command	Description
trigger.tsplinkin[N].clear() (on page 14-534)	Clears the event detector for a trigger
trigger.tsplinkin[N].edge (on page 14-535)	Indicates which trigger edge controls the trigger event detector for a trigger line
trigger.tsplinkin[N].overrun (on page 14-536)	Indicates if the event detector ignored an event while in the detected state
trigger.tsplinkin[N].wait() (on page 14-536)	Waits for a trigger
trigger.tsplinkout[N].assert() (on page 14-537)	Simulates the occurrence of the trigger and generates the corresponding trigger event
trigger.tsplinkout[N].logic (on page 14-538)	Defines the trigger output with output logic for a trigger line
trigger.tsplinkout[N].pulsewidth (on page 14-538)	Sets the length of time that the trigger line is asserted for output triggers
trigger.tsplinkout[N].release() (on page 14-539)	Releases a latched trigger on the given TSP-Link trigger line
trigger.tsplinkout[N].stimulus (on page 14-540)	Specifies the event that causes the synchronization line to assert a trigger
tsplink.group (on page 14-543)	The group number of the TSP-Link node
tsplink.initialize() (on page 14-543)	Initializes all instruments and enclosures in the TSP-Link system
tsplink.line[N].mode (on page 14-545)	Defines the trigger operation of a TSP-Link line as digital in or out or trigger in or out
tsplink.line[N].reset() (on page 14-545)	Resets some of the TSP-Link trigger attributes to their defaults
tsplink.line[N].state (on page 14-546)	Reads or writes the digital state of a TSP-Link synchronization line
tsplink.master (on page 14-547)	Reads the node number assigned to the master node
tsplink.node (on page 14-548)	Defines the node number
tsplink.readport() (on page 14-549)	Reads the TSP-Link synchronization lines as a digital I/O port
tsplink.state (on page 14-550)	Describes the TSP-Link online state
tsplink.writeport() (on page 14-551)	Writes to all TSP-Link synchronization lines as a digital I/O port

TSP-Link synchronization programming example

The programming example below illustrates how to set bit B1 of the TSP-Link digital I/O port high, and then read the entire port value:

```
tsplink.line[1].mode = tsplink.MODE_DIGITAL_OPEN_DRAIN
-- Set bit B1 high.
tsplink.line[1].state = 1
-- Read I/O port.
data = tsplink.readport()
print(data)
```

The output would be similar to:

```
7
```

To read bit B1 only:

```
-- To read bit B1 only
data = tsplink.line[1].state
print(data)
```

The output would be similar to:

```
tsplink.STATE_HIGH
```

Using DAQ6510 TSP-Link commands with other TSP-Link products

If you are connecting the DAQ6510 in a system with other TSP-Link products, be aware that some of the TSP-Link commands may be different.

Commands that are the same in all TSP-Link products:

- `tsplink.group`
- `tsplink.master`
- `tsplink.node`
- `tsplink.readport()`
- `tsplink.state`
- `tsplink.writeport()`

DAQ6510 TSP-Link command	Replaces this command in other TSP-Link products
<code>trigger.tsplinkin[N].clear()</code>	<code>tsplink.trigger[N].clear()</code>
<code>trigger.tsplinkin[N].edge</code> <code>trigger.tsplinkout[N].logic</code> <code>tsplink.line[N].mode</code>	<code>tsplink.trigger[N].mode</code>
<code>trigger.tsplinkin[N].overrun</code>	<code>tsplink.trigger[N].overrun</code>
<code>trigger.tsplinkin[N].wait()</code>	<code>tsplink.trigger[N].wait()</code>
<code>trigger.tsplinkout[N].assert()</code>	<code>tsplink.trigger[N].assert()</code>
<code>trigger.tsplinkout[N].pulsewidth</code>	<code>tsplink.trigger[N].pulsewidth</code>
<code>trigger.tsplinkout[N].release()</code>	<code>tsplink.trigger[N].release()</code>
<code>trigger.tsplinkout[N].stimulus</code>	<code>tsplink.trigger[N].stimulus</code>
<code>tsplink.initialize()</code>	<code>tsplink.reset()</code>
<code>tsplink.line[N].reset()</code>	<code>tsplink.trigger[N].reset()</code>
<code>tsplink.line[N].state</code>	<code>tsplink.readbit()</code> <code>tsplink.writebit()</code>
Not applicable	<code>tsplink.writeprotect</code>

TSP-Net

TSP-Net provides a simple socket-like programming interface to Test Script Processor (TSP) enabled instruments. Using the TSP-Net library, the DAQ6510 can control ethernet-enabled devices directly through its LAN port. This enables the DAQ6510 to communicate directly with a device that is not TSP-enabled without the use of a controlling computer.

Using TSP-Net library methods, you can transfer string data to and from a remote instrument, transfer and format data into Lua variables, and clear input buffers. The TSP-Net library is only accessible using commands from a remote command interface when you are using the TSP command language.

While you can use TSP-Net commands to communicate with any ethernet-enabled instrument, specific TSP-Net commands exist for TSP-enabled instruments to allow for support of features unique to the TSP scripting engine. These features include script downloads, reading buffer access, wait completion, and handling of TSP scripting engine prompts.

Using TSP-Net commands with TSP-enabled instruments, a DAQ6510 can download a script to another TSP-enabled instrument and have both instruments run scripts independently. The DAQ6510 can read the data from the remote instrument and either manipulate the data or send the data to a different remote instrument on the LAN.

You can use TSP-Net to connect to a computer; you can use a script on the instrument to transfer data directly to your computer hard drive.

With TSP-Net, you can simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the DAQ6510.

Using TSP-Net with any ethernet-enabled instrument

NOTE

Refer to [TSP command reference](#) (on page 14-1) for details about the commands presented in this section.

The DAQ6510 LAN port is auto-sensing (Auto-MDIX), so you can use either a LAN crossover cable or a LAN straight-through cable to connect directly from the DAQ6510 to an ethernet device or to a hub.

To set up communications to a remote ethernet-enabled instrument that is TSP® enabled:

1. Send the following command to configure TSP-Net to send an abort command when a connection to a TSP instrument is established:

```
tspnet.tsp.abortonconnect = 1
```

If the scripts are allowed to run, the connection is made, but the remote instrument may be busy.

2. Send the command:

```
connectionID = tspnet.connect(ipAddress)
```

Where:

- *connectionID* is the connection ID that is used as a handle in all other TSP-Net function calls.
- *ipAddress* is the IP address, entered as a string, of the remote instrument.

See [tspnet.connect\(\)](#) (on page 14-553) for additional detail.

To set up communications to a remote ethernet-enabled device that is not TSP enabled:

Send the command:

```
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

Where:

- *connectionID* is the connection ID that is used as a handle in all other *tspnet* function calls.
- *ipAddress* is the IP address, entered as a string, of the remote device.
- *portNumber* is the port number of the remote device.
- *initString* is the initialization string that is to be sent to *ipAddress*.

See [tspnet.connect\(\)](#) (on page 14-553) for additional detail.

To communicate to a remote ethernet device from the DAQ6510:

1. Connect to the remote device using one of the above procedures. If the DAQ6510 cannot make a connection to the remote device, it generates a timeout event. Use *tspnet.timeout* to set the timeout value. The default timeout value is 20 s.
2. Use *tspnet.write()* or *tspnet.execute()* to send strings to a remote device. If you use:

- `tspnet.write()`: Strings are sent to the device exactly as indicated, and you must supply any needed termination characters.
 - `tspnet.execute()`: The DAQ6510 appends termination characters to all strings that are sent. Use `tspnet.termination()` to specify the termination character.
3. To retrieve responses from the remote instrument, use `tspnet.read()`. The DAQ6510 suspends operation until the remote device responds or a timeout event is generated. To check if data is available from the remote instrument, use `tspnet.readavailable()`.
 4. Disconnect from the remote device using the `tspnet.disconnect()` function. Terminate all remote connections using `tspnet.reset()`.

Example script

The following example demonstrates how to connect to a remote device that is not TSP® enabled, and send and receive data from this device:

```
-- Set tspnet timeout to 5 s.
tspnet.timeout = 5
-- Establish connection to another device with IP address 192.168.1.51
-- at port 1394.
id_instr = tspnet.connect("192.168.1.51", 1394, "*rst\r\n")
-- Print the device ID from connect string.
print("ID is: ", id_instr)
-- Set the termination character to CRLF. You must do this
-- for each connection after the connection has been made.
tspnet.termination(id_instr, tspnet.TERM_CRLF)
-- Send the command string to the connected device.
tspnet.write(id_instr, "login admin\r\n")
-- Read the data available, then print it.
tspnet.write(id_instr, "*idn?\r\n")
print("instrument write/read returns: ", tspnet.read(id_instr))
-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

This example produces a return such as:

```
ID is:      1
instrument write/read returns:      SUCCESS: Logged in
instrument write/read returns:      KEITHLEY INSTRUMENTS,MODEL
                                     DAQ6510,04089762,1.6.3d
```

Remote instrument events

If the DAQ6510 is connected to a TSP-enabled instrument through TSP-Net, all events that occur on the remote instrument are transferred to the event log of the DAQ6510. The DAQ6510 indicates events from the remote instrument by prefacing these events with "Remote Error." For example, if the remote instrument generates event code 4909, "Reading buffer not found within device," the DAQ6510 generates the string "Remote Error: (4909) Reading buffer not found within device."

TSP-Net instrument commands: General device control

The following instrument commands provide general device control:

- [tspnet.clear\(\)](#) (on page 14-552)
- [tspnet.connect\(\)](#) (on page 14-553)
- [tspnet.disconnect\(\)](#) (on page 14-555)
- [tspnet.execute\(\)](#) (on page 14-556)
- [tspnet.idn\(\)](#) (on page 14-557)
- [tspnet.read\(\)](#) (on page 14-558)
- [tspnet.readavailable\(\)](#) (on page 14-559)
- [tspnet.reset\(\)](#) (on page 14-560)
- [tspnet.termination\(\)](#) (on page 14-561)
- [tspnet.timeout](#) (on page 14-562)
- [tspnet.write\(\)](#) (on page 14-567)

TSP-Net instrument commands: TSP-enabled device control

The following instrument commands provide TSP-enabled device control:

- [tspnet.tsp.abort\(\)](#) (on page 14-563)
- [tspnet.tsp.abortonconnect](#) (on page 14-564)
- [tspnet.tsp.rhtablecopy\(\)](#) (on page 14-565)
- [tspnet.tsp.runscript\(\)](#) (on page 14-566)

Example: Using tspnet commands

```
function telnetConnect(ipAddress, userName, password)
    -- Connect through Telnet to a computer.
    id = tspnet.connect(ipAddress, 23, "")
    -- Read the title and login prompt from the computer.
    print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
    print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
    -- Send the login name.
    tspnet.write(id, userName .. "\r\n")
    -- Read the login echo and password prompt from the computer.
    print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
    -- Send the password information.
    tspnet.write(id, password .. "\r\n")
    -- Read the telnet banner from the computer.
    print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
    print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
    print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
    print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
end

function test_tspnet()
    tspnet.reset()
    -- Connect to a computer using Telnet.
    telnetConnect("192.0.2.1", "my_username", "my_password")
    -- Read the prompt back from the computer.
    print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
    -- Change directory and read the prompt back from the computer.
    tspnet.write(id, "cd c:\\\\r\n")
    print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
    -- Make a directory and read the prompt back from the computer.
    tspnet.write(id, "mkdir TEST_TSP\r\n")
    print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
    -- Change to the newly created directory.
    tspnet.write(id, "cd c:\\\\TEST_TSP\r\n")
    print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
    -- if you have data print it to the file.
    -- 11.2 is an example of data collected.
    cmd = "echo " .. string.format("%g", 11.2) .. " >> datafile.dat\r\n"
    tspnet.write(id, cmd)
    print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
    tspnet.disconnect(id)
end
test_tspnet()
```

Maintenance

In this section:

Introduction	10-1
Line fuse replacement.....	10-1
Line voltage verification.....	10-3
Input fuse replacement.....	10-4
Lithium battery.....	10-5
Front-panel display.....	10-5
Upgrading the firmware.....	10-7

Introduction

This section describes routine maintenance of the instrument that an operator can perform.

Line fuse replacement

A fuse on the DAQ6510 rear panel protects the power-line input of the instrument. The following instructions describe how to replace the fuse. You do not need to return your instrument for service if the fuse is damaged.

WARNING

Disconnect the line cord at the rear panel and remove all test leads connected to the instrument before replacing a line fuse. Failure to do so could expose the operator to hazardous voltages that could result in personal injury or death.

Use only the correct fuse type. Failure to do so could result in injury, death, or instrument damage.

Complete the following steps to replace the line fuse:

1. Power off the instrument.
2. Remove all test leads connected to the instrument.
3. Remove the line cord.
4. Locate the fuse case, which is above the AC receptacle, as shown in the figure below.

Figure 150: DAQ6510 AC receptacle and line fuse location

CAUTION

Make sure to note the position of the voltage value closest to the AC power cord receptacle on the fuse drawer before you begin.

5. Squeeze the tabs on the fuse case and remove the fuse case from the AC power module.
6. Remove the fuse from the fuse case
7. Replace the fuse.
8. Make sure that the position of the fuse case is correct, then push the fuse case back into the module.

If the fuse continues to become damaged, a circuit malfunction exists and must be corrected. Return the instrument to Keithley Instruments for repair.

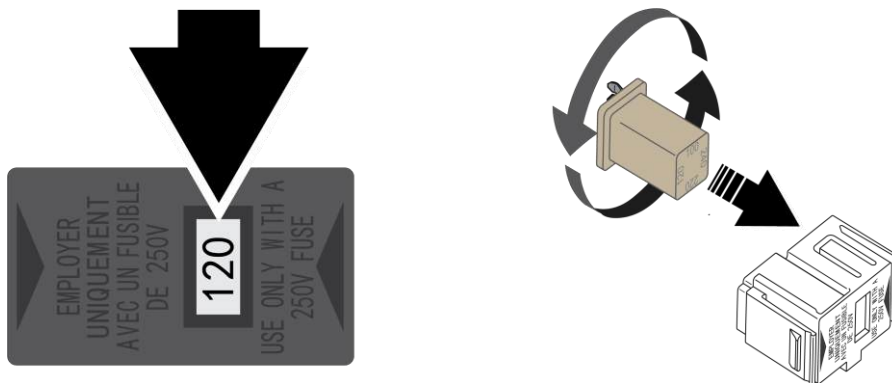
Line voltage verification

The fuse is set to the expected voltage at the factory. Make sure that the correct line voltage is displayed on the power module. If you need to select another line voltage, follow the below procedure.

To change the fuse orientation:

1. Make sure the POWER is off.
2. Remove all test leads connected to the instrument.
3. Remove the power cord.
4. Squeeze the tabs on the fuse case and remove the fuse case from the power module.
5. Remove the fuse from the fuse case.
6. Rotate the fuse so the proper voltage on the fuse shows in the fuse case when installed.
7. Install the fuse case in the power module.

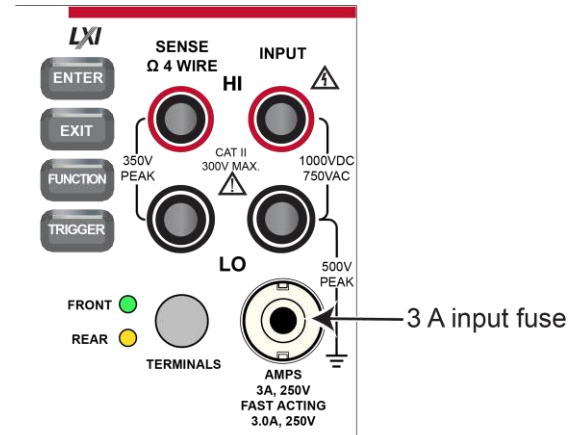
Figure 151: Verifying the line voltage and changing the fuse orientation



Input fuse replacement

The input line from the AMPS connector on the front panel is protected by a 3 A, 250 V fuse.

Figure 152: DAQ6510 front-panel current-input fuse location



⚠ WARNING

Make sure the instrument is disconnected from the power line and other equipment before checking or replacing a current-input fuse. 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.

CAUTION

For continued protection against fire or instrument damage, only replace fuse with the type and rating listed. If the instrument repeatedly damages fuses, locate and correct the cause of the problem before replacing the fuse.

To replace a current-input fuse:

1. Turn off the power to the instrument.
2. Disconnect the power line and test leads.
3. From the front panel, gently push in the AMPS fuse holder and rotate it one-quarter turn counter-clockwise.
4. Remove the fuse and replace it with the same type.
5. Install the new fuse by reversing the procedure above.

NOTE

If the fuse continues to become damaged, a circuit malfunction exists and must be corrected. Return the instrument to Keithley Instruments for repair.

Lithium battery

The DAQ6510 contains a CR2032 cell (LiMnO₂) battery. Perchlorate material may require special handling. See [Hazardous waste - perchlorate \(dtsc.ca.gov/hazardouswaste/perchlorate\)](https://dtsc.ca.gov/hazardouswaste/perchlorate).

This battery is not replaceable by the user.

Front-panel display

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

Cleaning the front-panel display

If you need to clean the front-panel LCD touchscreen display, use a soft dry cloth.

CAUTION

Do not use liquids to clean the display.

Abnormal display operation

If the display area is pushed hard during operation, you may see abnormal display operation. To restore normal operation, turn the instrument off and then back on.

Removing ghost images or contrast irregularities

If the display has been operating for a long time with the same display patterns, the display patterns may remain on the screen as ghost images and a slight contrast irregularity may appear. Note that if this occurs, it does not adversely affect the performance reliability of the display.

To regain normal operation, stop using the front-panel display for some time. You can turn off the front-panel display while continuing operation using remote commands and the virtual front panel.

To turn off the front-panel display using a SCPI command:

Send the command:

```
DISPlay:LIgHT:STATe OFF
```

To turn off the front-panel display using a TSP command:

Send the command:

```
display.lightstate = display.STATE_LCD_OFF
```

Upgrading the firmware

To upgrade the DAQ6510 firmware, you load an upgrade file into the instrument. You can load the file from the front-panel USB port using either a remote interface or the front panel of the instrument. If you are using Test Script Builder (TSB), you can upgrade the firmware from TSB using a file saved to the computer on which TSB is running.

The firmware file must be in the root subdirectory of the flash drive and must be the only firmware file in that location.

During the upgrade process, the instrument verifies that the version you are loading is newer than what is on the instrument. If the version is older or at the same revision level, no changes are made. If you have a communications accessory card (KTTI-GPIB, KTTI-TSP, or KTTI-RS232) installed in the instrument, the firmware on the card is also upgraded.

If you want to return to a previous version or reload the present version of the firmware, select **Downgrade to Older**. This forces the instrument to load the firmware regardless of the version.

The upgrade process normally takes about five minutes.

Upgrade files are available on tek.com/keithley (tek.com/products/keithley).

CAUTION

Disconnect the input terminals before you upgrade or downgrade.

Do not remove power from the instrument or remove the USB flash drive while an upgrade or downgrade is in progress. Wait until the instrument completes the procedure and shows the opening display.

Do not initialize or reset TSP-Link before starting the upgrade.

Before starting the upgrade, turn the instrument power off, wait a few seconds, then turn the instrument power on.

From the front panel

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

NOTE

The firmware file must be in the root subdirectory of the flash drive and must be the only firmware file in that location. You can upgrade or downgrade the firmware from the front panel or from the virtual front panel. Refer to [Using the DAQ6510 virtual front panel](#) (on page 2-37) for information.

From the front panel:

1. Copy the firmware file (.upg file) to a USB flash drive.
2. Verify that the firmware file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect any terminals that are attached to the instrument.
4. Turn the instrument power off. Wait a few seconds.
5. Turn the instrument power on.
6. Insert the flash drive into the USB port on the front panel of the instrument.
7. From the instrument front panel, press the **MENU** key.
8. Under System, select **Info/Manage**.
9. Choose an upgrade option:
 - To upgrade to a newer version of firmware, select **Upgrade to New**.
 - To return to a previous version of firmware, select **Downgrade to Older**.
10. If the instrument is controlled remotely, a message is displayed. Select **Yes** to continue.
11. When the upgrade is complete, reboot the instrument.

A message is displayed while the upgrade is in progress.

Using SCPI

There are no SCPI commands that you can use to upgrade the firmware. To upgrade the firmware, you must either use the front panel, virtual front panel, or switch the command set to TSP.

To use the front panel to upgrade the firmware, see [From the front panel](#) (on page 10-8).

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

If you need to upgrade the firmware from a remote interface and you are using a SCPI command set:

1. Copy the firmware upgrade file to a USB flash drive.
2. Verify that the upgrade file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect the input and output terminals that are attached to the instrument.
4. Power on the instrument.
5. Change the command set to TSP by sending the command:
*LANG TSP
6. Turn the instrument off and then turn it on again.
7. Insert the flash drive into the USB port on the front panel of the instrument.
8. Choose an upgrade option:
 - To upgrade to a newer version of firmware, send:
upgrade.unit()
 - To return to a previous version of firmware, send:
upgrade.previous()
9. After completion of the upgrade, turn the instrument off and then turn it on again.
10. To return to the SCPI command set, send the command:
*LANG SCPI
11. Turn the instrument off and then turn it on again.

A message is displayed on the front panel of the instrument while the upgrade is in process.

Using TSP

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

Using TSP over a remote interface:

1. Copy the firmware upgrade file to a USB flash drive.
2. Verify that the upgrade file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect the input and output terminals that are attached to the instrument.
4. Power on the instrument.
5. Insert the flash drive into the USB port on the front panel of the instrument.
6. Choose an upgrade option:
 - To upgrade to a newer version of firmware, send:
`upgrade.unit()`
 - To return to a previous version of firmware, send:
`upgrade.previous()`
7. After completion of the upgrade, reboot the instrument.

A message is displayed on the front panel of the instrument while the upgrade is in progress.

Using TSB

CAUTION

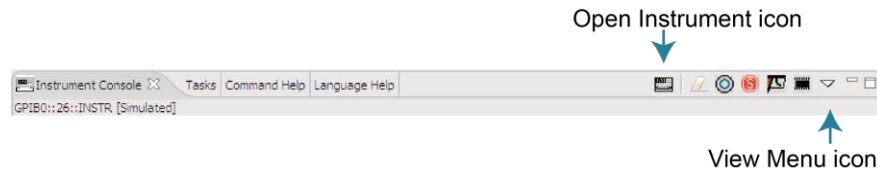
Do not turn off power or remove the USB flash drive until the upgrade process is complete.

You can use Test Script Builder (TSB) to upgrade the firmware of your instrument.

To upgrade the firmware using Test Script Builder:

1. Disconnect the input and output terminals that are attached to the instrument.
2. Start Test Script Builder.
3. On the Instrument Console toolbar, select the **Open Instrument** icon.

Figure 153: TSB Instrument Console toolbar



4. Choose your communications interface from the Select Instrument dialog box. See the [Connecting an instrument in TSB](#) (on page 13-42) for details on opening communications.
5. On the Instrument Console toolbar, select the **View Menu** icon. Select **Instrument**, then select **Flash**.
6. From the Select a Firmware Image File dialog box, use the browser to select the file name of the new firmware or enter the path and file name.
7. Choose an upgrade option:
 - If you are upgrading the firmware, replace the existing firmware with a newer version of firmware.
 - If you are downgrading the firmware, replace the existing firmware with an older version of firmware or repair the same version.
8. Select **OK**. A Progress Information bar is displayed on the instrument during the update.
9. Wait until the instrument indicates that the firmware upgrade is complete (TSB may indicate that the upgrade is complete before it is finalized on the instrument).
10. Reboot the instrument.

Introduction to SCPI commands

In this section:

Introduction to SCPI	11-1
SCPI command formatting	11-4
Using the SCPI command reference.....	11-7

Introduction to SCPI

The Standard Commands for Programmable Instruments (SCPI) standard is a syntax and set of commands that are used to control test and measurement devices.

The following information describes some basic SCPI command information and how SCPI is used with the DAQ6510 and presented in the DAQ6510 documentation.

This section also contains general information about using SCPI.

Command execution rules

Command execution rules are as follows:

- Commands execute in the order that they are presented in the command message.
- An invalid command generates an event message and is not executed.
- Valid commands that precede an invalid command in a command message are executed.
- Valid commands that follow an invalid command in a command message are ignored.

Command messages

A command message is made up of one or more command words sent by the controller to the instrument.

SCPI commands contain several command words that are structured to create command messages. The command words are separated by colons (:). For example, to configure an ethernet connection, the command words are:

```
:SYSTem:COMMunication:LAN:CONFigure
```

Many commands have query options. If there is a query option, it is created by adding a question mark (?) to the command. For example, to query the present ethernet settings, send:

```
:SYSTem:COMMunication:LAN:CONFigure?
```

Commands often take parameters. Parameters follow the command words and a space. For example, to set the instrument to automatically detect the ethernet settings, send:

```
:SYSTem:COMMunication:LAN:CONFigure "AUTO"
```

SCPI can also use common commands, which consist of an asterisk (*) followed by three or four letters. For example, you can reset the instrument by sending the following command:

```
*RST
```

The examples above show commands that are sent individually. You can also group command messages when you send them to the instrument. To group a set of commands, separate them with semicolons and include a colon before each command (unless it starts with an *).

For example, to reset the instrument, enable relative offset for the current function, and set a relative offset of 0.5 for the current function, send the command:

```
*RST; :SENSe:CURRent:REL:STAT ON; :SENSe:CURRent:RELative 0.5
```

If commands are not combined, the colon (:) at the beginning of a command is optional. For example, the following commands are equivalent:

```
:SENSe:CURRent:REL:STAT ON  
SENSe:CURRent:REL:STAT ON
```

If the next command in a multiple command message is on the same path, you do not need to send the colon or the path to reset the path parsing of the command. For example, the following examples for returning the system time and system version are equivalent:

```
:SYST:TIME?;:SYST:VERSION?  
:SYST:TIME?;VERSION?
```

Both return:

```
1569191196;1996.0
```

You can also do multiple queries in a single command message with or without resetting the path. For example, to query for the current relative offset and state, you can send:

```
:SENSe:CURRent:RELative?; :SENSe:CURRent:REL:STAT?
```

You can also send:

```
SENSe:CURRent:RELative?; rel:STAT?
```

Each new command message resets the parser path as if it was sent with the leading colon. The output for both queries is:

```
0.5;1
```

A command string sent to the instrument must terminate with a <new line> character. The IEEE-488.2 EOI (end-or-identify) message is interpreted as a <new line> character and can be used to terminate a command string in place of a <new line> character. A <carriage return> followed by a <new line> is also accepted. Command string termination will always reset the present SCPI command path to the root level.

SCPI command formatting

This section describes the formatting that this manual uses when discussing SCPI commands.

SCPI command short and long forms

This documentation shows SCPI commands with both uppercase and lowercase letters. The uppercase letters are the required elements of a command. The lowercase letters are optional. If you choose to include the letters that are shown in lowercase letters, you must include all of them.

When you send a command to the instrument, letter case is not important. You can mix uppercase and lowercase letters in program messages.

For example, you can send the command `SENSe:COUNT?` in any of the following formats:

```
SENSe:COUNT?  
sense:count?  
SENS:COUN?  
Sens:Coun?
```

Optional command words

If a command word is enclosed in brackets (`[ ]`), the command word is optional. Do not include the brackets if you send the optional command word to the instrument.

For example, you can send the command `:SYSTem:BEEPer[:IMMediate] <n1>, <n2>` in any of the following formats:

```
:SYSTem:BEEPer:IMMediate 500, 1  
:SYSTem:BEEPer 500, 1  
:SYST:BEEP:IMMediate 500, 1  
:SYST:BEEP 500, 1
```

MINimum, MAXimum, and DEFault

You can use `MINimum`, `MAXimum`, or `DEFault` instead of a parameter for some commands.

For example, you can set the parameter for the command

`[[:SENSe[1]]:RESistance:NPLCycles` to the minimum, maximum, or default value. To set NPLC to the minimum value, you can send either of these commands:

```
:SENSe1:RESistance:NPLCycles MINimum  
:SENS:RES:NPLC MIN
```


Queries

SCPI queries have a question mark (?) after the command. You can use the query to determine the present value of the parameters of the command or to get information from the instrument.

For example, to determine what the present setting for NPLC is, you can send:

```
:SENSel:RESistance:NPLCycles?
```

This query returns the present setting.

If the command has `MINimum`, `MAXimum`, and `DEFault` options, you can use the query command to determine what the minimum, maximum, and default values are. In these queries, the ? is placed before the `MINimum`, `MAXimum`, or `DEFault` parameter. For example, to determine the default value for NPLC, you can send:

```
:SENSel:RESistance:NPLCycles? DEFault
```

If you send two query commands without reading the response from the first, and then attempt to read the second response, you may receive some data from the first response followed by the complete second response. To avoid this, do not send a query command without reading the response. When you cannot avoid this situation, send a device clear before sending the second query command.

When you query a Boolean option, the instrument returns a 0 or 1, even if you sent `OFF` or `ON` when you originally sent the command.

SCPI parameters

The parameters of the SCPI commands are shown in angle brackets (< >). For example:

```
:SYSTem:BEEPer[:IMMediate] <frequency>, <duration>
```

The type of information that you can use to replace <frequency> and <duration> is defined in the Usage section of the command description. For this example, the Usage is:

<frequency>	The frequency of the beep (20 Hz to 8000 Hz)
<duration>	The amount of time to play the tone (0.001 s to 100 s)

For this example, you can generate an audible sound by sending:

```
:SYSTem:BEEPer 500, 1
```

Do not include the angle brackets when sending the command.

Sending strings

If you are sending a string, it must begin and end with matching quotes (either single quotes or double quotes). To include a quote character as part of the string, type it twice with no characters in between.

Channel naming

In the DAQ6510, channels are named using the slot number and the channel number. The slot number is the number of the slot on the rear panel where the card is installed. The channel assignment is formatted as `SCH`, where:

- `S` is the slot number
- `CH` is a two-digit channel number

For example:

- `101` = slot 1, channel 1
- `210` = slot 2, channel 10

If you are sending commands from a remote interface, you include the three-digit channel assignment in the channel list parameter for the commands.

In SCPI commands, the first channel number in a command is prefaced by `@`. To designate multiple individual channels, separate the channels with commas. In the following example, the command will close channels 1 and 3 on the card in slot 1.

```
ROUT:CLOS (@101, 103)
```

To designate a range of channels, separate the first and last channel number with a colon. The range can be from lowest to highest numbered channel or highest to lowest. The range can include channels from both slots. The following example sets DMM settings on a range of twenty channels and then creates a scan with those channels using the DC voltage function.

```
SENS:FUNC "VOLT", (@101:120)  
SENS:VOLT:NPLC 0.1, (@101:120)  
ROUT:SCAN:CRE (@101:120)  
INIT
```

Some commands allow you to set all channels in a slot or all channels in the instrument. In this case, you can send `SLOT1` to set all channels in slot 1; `SLOT2` for all channels in slot 2, or `ALLSLOTS` for all channels. For example, to set all channels on slot 1 to measure voltage, you can send:

```
SENS:FUNC "VOLT", (@SLOT1)
```

When you send a range that includes both slots, SLOT1, SLOT2, or ALLSLOTS to a slot that contains channels that cannot accept that setting, the instrument generates an error, but the change is made to all channels for which the setting is allowed.

Using the SCPI command reference

The SCPI command reference contains detailed descriptions of each of the SCPI commands that you can use to control your instrument. Each command description is broken into several standard subsections. The following figure shows an example of a command description.

Figure 154: SCPI command description example

:EXAMple:COMManD:STATe
This command is an example of a typical SCPI command that turns an instrument feature on or off.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1 (ON)

Usage

```
:EXAMple:COMManD:STATe <state>  
:EXAMple:COMManD:STATe?
```

<state>

Disable the example feature: 0 or OFF
Enable the example feature: 1 or ON

Details
This command is an example of a typical SCPI command that enables or disables a feature.

Example

:EXAMple:COMManD:STATe ON

Turn the example feature on.

Also see
[:EXAMple:COMManD:UNIT](#) (on page 6-100)

Each command listing is divided into five subsections that contain information about the command:

- Command name and summary table
- Usage
- Details
- Example
- Also see

The content of each of these subsections is described in the following topics.

Command name and summary table

Each instrument command description starts with the command name, followed by a table with relevant information for each command. Definitions for the numbered items are listed following the figure.

Figure 155: SCPI command name and summary table

The diagram illustrates the structure of an SCPI command listing. It shows a command name, a descriptive sentence, and a summary table. Numbered callouts point to specific elements: 1 points to the command name, 2 points to the descriptive sentence, 3 points to the 'Affected by' column of the table, 4 points to the 'Where saved' column, and 5 points to the 'Default value' column.

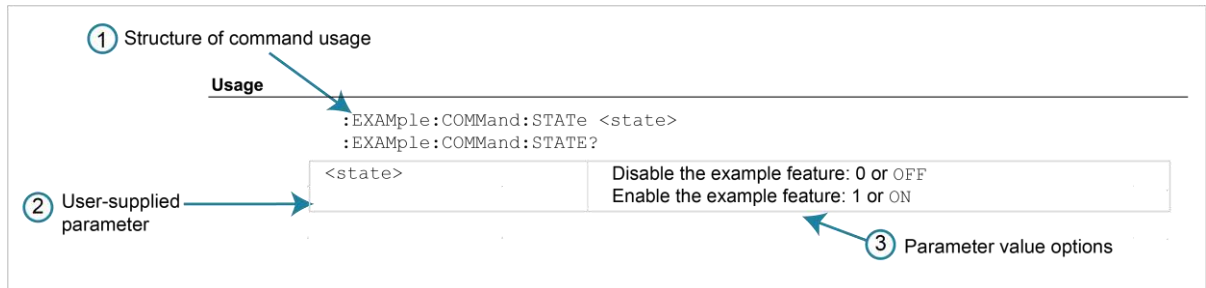
:EXAMple:COMManD:STATe			
This command is an example of a typical SCPI command that turns an instrument feature on or off.			
Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1 (ON)

- 1 **Instrument command name.** Signals the beginning of the command description and is followed by a brief description of what the command does.
- 2 **Type of command.** Options are:
 - **Command only.** There is a command but no query option for this command.
 - **Command and query.** The command has both a command and query form.
 - **Query only.** This command is a query.
- 3 **Affected by.** Commands or actions that have a direct effect on the instrument command.
 - **Recall settings.** If you send *RCL to recall the system settings, this setting is changed to the saved value.
 - **Instrument reset.** When you reset the instrument, this command is reset to its default value. Reset can be done from the front panel or when you send *RST.
 - **Power cycle.** When you power cycle the instrument, this command is reset to its default value.
 - **Measure configuration list.** If you recall a measure configuration list, this setting changes to the stored setting.
- 4 **Where saved.** Indicates where the command settings reside once they are used on an instrument. Options include:
 - **Not saved.** Command is not saved and must be sent each time you use it.
 - **Nonvolatile memory.** The command is stored in a storage area in the instrument where information is saved even when the instrument is turned off.
 - **Save settings.** This command is saved when you send the *SAV command.
 - **Measure configuration list.** This command is stored in measure configuration lists.
- 5 **Default value:** Lists the default value for the command. The parameter values are defined in the Usage or Details sections of the command description.

Command usage

The Usage section of the remote command listing shows how to properly structure the command. Each line in the Usage section is a separate variation of the command usage; all possible command usage options are shown here.

Figure 156: SCPI command description usage identification



1. **Structure of command usage:** Shows the organization of the parts of the command.
2. **User-supplied parameters:** Indicated by angle brackets (< >).

NOTE

Some commands have optional parameters. Optional parameters are presented on separate lines in the Usage section, presented in the required order with each valid permutation of optional parameters. For example:

```
:SYSTem:COMMunication:LAN:CONFigure AUTO
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress, NETmask
:SYSTem:COMMunication:LAN:CONFigure MANual, IPaddress, NETmask,
GATeway
:SYSTem:COMMunication:LAN:CONFigure?
```

3. **Parameter value options:** Descriptions of the options that are available for the parameter.

Command details

This section lists additional information you need to know to successfully use the command.

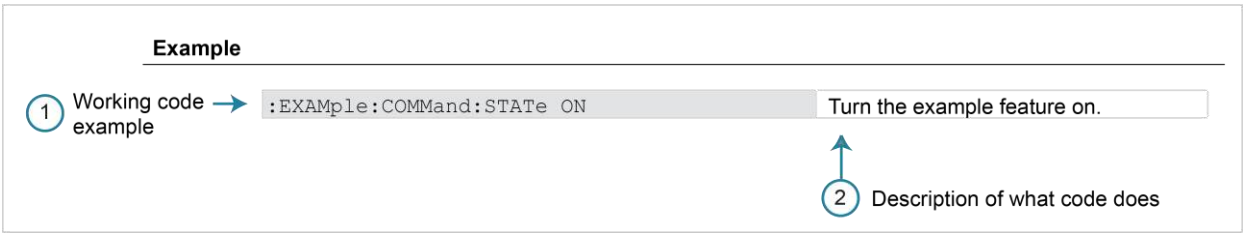
Figure 157: Details section of command listing



Example section

The Example section of the command description shows some simple examples of how the command can be used.

Figure 158: SCPI command description code examples



1. Example code that you can copy from this table and paste into your own application. Examples are generally shown using the short forms of the commands.
2. Description of the code and what it does. This may also contain the output of the code.

Related commands list

The **Also see** section of the remote command description provides links to commands that are related to the command.

Figure 159: SCPI related commands list example



SCPI command reference

In this section:

*RCL	12-1
*SAV	12-2
:FETCh?	12-3
:MEASure:DIGitize?	12-6
:MEASure?	12-8
:READ:DIGitize?	12-11
:READ?	12-14
CALCulate subsystem	12-16
CALibration subsystem	12-37
DIGital subsystem	12-43
DISPlay subsystem	12-48
FORMat subsystem	12-56
ROUTe subsystem	12-59
SCRipt subsystem	12-113
SENSe1 subsystem	12-115
STATus subsystem	12-189
SYSTem subsystem	12-196
TRACe subsystem	12-229
TRIGger subsystem	12-268

*RCL

This command returns the instrument to the setup that was stored in a specified memory location.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*RCL <n>

<n>

An integer from 0 to 4 that represents the saved setup

Details

Restores the state of the instrument from a copy of user-saved settings that are stored in setup memory. The settings are saved using the *SAV command.

If you view the user-saved settings from the front panel of the instrument, these are stored as scripts named Setup0<n>.

Example

*RCL 3

Restores the settings stored in memory location 3.

Also see

[Saving setups](#) (on page 4-92)
[*SAV](#) (on page 12-2)

***SAV**

This command saves the present instrument settings as a user-saved setup.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Nonvolatile memory	Not applicable

Usage

*SAV <n>

<n>	An integer from 0 to 4

Details

Save the present instrument settings as a user-saved setup. You can restore the settings with the *RCL command.

Most commands that are affected by *RST can be saved with the *SAV command.

You can save up to five user-saved setups. Any settings that had been stored previously as <n> are overwritten.

If you view the user-saved setups from the front panel of the instrument, they are stored as scripts named Setup0<n>.

NOTE

Settings made on the Graph and Histogram tabs are not saved as part of a saved setup. To record graph settings, you can press HOME and ENTER to save an image of the settings with the screen capture feature. Refer to [Save screen captures to a USB flash drive](#) (on page 3-79) for additional information.

Example

*SAV 2	Saves the instrument settings in memory location 2.

Also see

[Saving setups](#) (on page 4-92)
[*RCL](#) (on page 12-1)

:FETCh?

This command requests the latest reading from a reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:FETCh?  
:FETCh? "<bufferName>"  
:FETCh? "<bufferName>", <bufferElements>
```

<bufferName>	The name of the buffer where the reading is stored; if nothing is specified, defbuffer1 is used
<bufferElements>	See Details ; default is READing

Details

This command requests the last available reading from a reading buffer. If you send this command more than once and there are no new readings, the returned values are the same. If the reading buffer is empty, an error is returned.

NOTE

To change the number of digits returned in a remote command reading, use the :FORMat:ASCIi:PRECision command.

You can send `:FETCh?` while a trigger model is running.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
CHANnel	The channel from which the data was acquired
DATE	The date when the data point was measured; the buffer style must be set to the style standard or full to use this option
EXTRa	Returns an additional value (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAFORMatted	Returns the measurement and the unit of measure of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAUNIT	Returns the units of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
FORMatted	The measured value as it appears on the front panel
FRACTional	The fractional seconds when the data point was measured
READing	The measurement reading
RELative	The relative time when the data point was measured
SEConds	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
STATus	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time when the data point was measured
TSTamp	The timestamp when the data point was measured
UNIT	The unit of measure of the measurement

The output of `:FETCh?` is affected by the data format selected by `:FORMat[:DATA]`. If you set `FORMat[:DATA]` to `REAL` or `SREAL`, you will have fewer options for buffer elements. The only buffer elements available are `READing`, `RELative`, and `EXTRa`. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

The `STATus` buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high
0x0100	STAT_START_GROUP	256	First reading in a group

Example

```
FETCh? "defbuffer1", DATE, READ
```

Retrieve the date and measurement value for the most recent data captured in `defbuffer1`.

Example output:

```
03/21/2019,-1.375422E-11
```

Also see

[:FORMat:DATA](#) (on page 12-58)
[:INITiate:IMMediate](#) (on page 12-60)
[:MEASure?](#) (on page 12-8)
[:MEASure:DIGitize?](#) (on page 12-6)
[:READ?](#) (on page 12-14)
[:READ:DIGitize?](#) (on page 12-11)
[:TRACe:DATA?](#) (on page 12-237)
[:TRACe:TRIGger](#) (on page 12-260)
[:TRACe:TRIGger:DIGitize](#) (on page 12-261)

:MEASure:DIGitize?

This command makes a digitize measurement, places it in a reading buffer, and returns the reading.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:MEASure:DIGitize?  
:MEASure:DIGitize:<function>?  
:MEASure:DIGitize:<function>? "<bufferName>"  
:MEASure:DIGitize:<function>? "<bufferName>", <bufferElements>  
:MEASure:DIGitize? "<bufferName>"  
:MEASure:DIGitize? "<bufferName>", <bufferElements>
```

<function>	The function to use for the measurement: ■ Voltage: VOLTage ■ Current: CURRent If no function is defined, the presently selected one is used
<bufferName>	The name of the buffer where the reading is stored; if nothing is specified, defbuffer1 is used
<bufferElements>	See Details

Details

This command makes a digitize measurement using the specified function and stores the reading in a reading buffer. Sending this command changes the measurement function to the one specified by <function>. This function remains selected after the measurement is complete.

If you do not define the function parameter, the instrument uses the presently selected function. If a digitize function is presently selected, an error is generated.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

To get multiple readings, use the :TRACe:DATA? command.

```
:MEASure:DIGitize? performs the same function as READ:DIGitize?.  
:MEASure:DIGitize:<function>? performs the same function as sending  
:SENSe:DIGitize:FUNCTION "<function>", then READ?.
```

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.

Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
CHANnel	The channel from which the data was acquired
DATE	The date when the data point was measured; the buffer style must be set to the style standard or full to use this option
EXTRa	Returns an additional value (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAFORMatted	Returns the measurement and the unit of measure of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAUNIT	Returns the units of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
FORMatted	The measured value as it appears on the front panel
FRACTIONal	The fractional seconds when the data point was measured
READing	The measurement reading
RELative	The relative time when the data point was measured
SEConds	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
STATus	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time when the data point was measured
TSTamp	The timestamp when the data point was measured
UNIT	The unit of measure of the measurement

The output of `:MEASure:DIGitize?` is affected by the data format selected by `:FORMat[:DATA]`. If you set `FORMat[:DATA]` to `REAL` or `SREAL`, you will have fewer options for buffer elements. The only buffer elements available are `READing`, `RELative`, and `EXTRa`. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

The `STATus` buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high

0x0100	STAT_START_GROUP	256	First reading in a group

Example

TRACe:MAKE "voltDigitizeBuffer", 10000
MEAS:DIG:VOLT? "voltDigitizeBuffer", FORM, DATE, READ

Create a buffer named `voltMeasBuffer`. Make a digitize voltage reading and store it in the buffer `voltMeasBuffer` and return the formatted reading, the date, and the reading elements from the buffer.
Example output:
-00.0024 mV, 05/16/2018, -2.384862E-06

Also see

[:READ:DIGitize?](#) (on page 12-11)
[:TRACe:DATA?](#) (on page 12-237)

:MEASure?

This command makes measurements, places them in a reading buffer, and returns the last reading.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:MEASure?
:MEASure:<function>?
:MEASure:<function>? "<bufferName>"
:MEASure:<function>? "<bufferName>", <bufferElements>
:MEASure? "<bufferName>"
:MEASure? "<bufferName>", <bufferElements>

<function>	The function to which the setting applies; see Functions

<bufferName>	The name of the buffer where the reading is stored; if nothing is specified, <code>defbuffer1</code> is used
<bufferElements>	See Details

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command makes a measurement using the specified function and stores the reading in a reading buffer.

If you do not define the function parameter, the instrument uses the presently selected measure function. If a digitize function is presently selected, an error is generated.

This query makes the number of readings specified by [:SENSe[1]]:COUNT. When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

To get multiple readings, use the :TRACe:DATA? command.

Sending this command changes the measurement function to the one specified by <function>. This function remains selected after the measurement is complete.

:MEASure? performs the same function as READ?.

:MEASure:<function>? performs the same function as sending :SENSe:FUNCTION, then READ?.

NOTE

To change the number of digits returned in a remote command reading, use the :FORMat:ASCIi:PRECision command.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for <bufferElements> are described in the following table.

Option	Description
CHANnel	The channel from which the data was acquired
DATE	The date when the data point was measured; the buffer style must be set to the style standard or full to use this option
EXTRa	Returns an additional value (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAFORMatted	Returns the measurement and the unit of measure of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAUNIT	Returns the units of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
FORMatted	The measured value as it appears on the front panel
FRActional	The fractional seconds when the data point was measured
READing	The measurement reading
RELative	The relative time when the data point was measured
SEConds	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
STATus	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time when the data point was measured
TSTamp	The timestamp when the data point was measured
UNIT	The unit of measure of the measurement

The output of :MEASure? is affected by the data format selected by :FORMat[:DATA]. If you set FORMat[:DATA] to REAL or SREAL, you will have fewer options for buffer elements. The only buffer elements available are READing, RELative, and EXTRa. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

The STATus buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high

0x0100	STAT_START_GROUP	256	First reading in a group

Example

```
TRACe:MAKE "voltMeasBuffer", 10000
MEAS:VOLT? "voltMeasBuffer", FORM, DATE, READ
```

Create a buffer named `voltMeasBuffer`. Make a voltage measurement and store it in the buffer `voltMeasBuffer` and return the formatted reading, the date, and the reading elements from the buffer.
Example output:
`-00.0024 mV, 05/16/2018, -2.384862E-06`

Also see

- [:FORMat\[:DATA\]](#) (on page 12-58)
- [:READ?](#) (on page 12-14)
- [\[:SENSe\[1\]\]:FUNction\[:ON\]](#) (on page 12-189)
- [:TRACe:DATA?](#) (on page 12-237)

:READ:DIGitize?

This command makes a digitize measurement, places it in a reading buffer, and returns the latest reading.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:READ:DIGitize?
:READ:DIGitize? "<bufferName>"
:READ:DIGitize? "<bufferName>", <bufferElements>
```

<bufferName>	The name of the buffer where the reading is stored; if nothing is specified, <code>defbuffer1</code> is used
<bufferElements>	See Details ; if nothing is specified, <code>READing</code> is used

Details

You must set the instrument to a digitize function before sending this command.

This query makes the number of readings specified by `[ :SENSe[1] ] :DIGitize:COUNT`. If multiple readings are made, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command. To get multiple readings, use the `:TRACe:DATA?` command.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for `<bufferElements>` are described in the following table.

Option	Description
CHANnel	The channel from which the data was acquired
DATE	The date when the data point was measured; the buffer style must be set to the style standard or full to use this option
EXTRa	Returns an additional value (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAFORMatted	Returns the measurement and the unit of measure of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAUNIT	Returns the units of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
FORMatted	The measured value as it appears on the front panel
FRACTIONal	The fractional seconds when the data point was measured
READing	The measurement reading
RELative	The relative time when the data point was measured
SEConds	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
STATus	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time when the data point was measured
TSTamp	The timestamp when the data point was measured
UNIT	The unit of measure of the measurement

The output of `:READ:DIG?` is affected by the data format selected by `:FORMat[:DATA]`. If you set `FORMat[:DATA]` to `REAL` or `SREAL`, you will have fewer options for buffer elements. The only buffer elements available are `READing`, `RELative`, and `EXTRa`. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

The `STATus` buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high
0x0100	STAT_START_GROUP	256	First reading in a group

Example

```
*RST
:TRACe:MAKE "voltDigBuffer", 10000
:DIG:FUNC "VOLTage"
:SENS:DIG:COUN 100
:READ:DIG? "voltDigBuffer", FORM, DATE, READ
:TRAC:DATA? 95, 100, "voltDigBuffer"
```

Create a buffer named `voltDigBuffer`. Make a digitize measurement, store it in the buffer `voltDigBuffer`, and return the formatted readings, date, and reading buffer elements for the last reading stored in `voltDigBuffer`, then return readings 95 to 100.

Example output is:

```
+04.963 V,09/26/2018,4.962954E+00
4.961211E+00,4.961695E+00,4.961889E+00,4.961985E+00,4.962276E+00,4.962954E+00
```

Also see

[:FETCh?](#) (on page 12-3)
[\[:SENSe\[1\]\]:DIGitize:COUNt](#) (on page 12-187)
[\[:SENSe\[1\]\]:DIGitize:FUNCTION\[:ON\]](#) (on page 12-188)
[:TRACe:DATA?](#) (on page 12-237)
[:TRACe:MAKE](#) (on page 12-243)
[:TRACe:TRIGger:DIGitize](#) (on page 12-261)

:READ?

This command makes measurements, places them in a reading buffer, and returns the last reading.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:READ?  
:READ? "<bufferName>"  
:READ? "<bufferName>", <bufferElements>
```

<bufferName>	The name of the buffer where the reading is stored; if nothing is specified, defbuffer1 is used
<bufferElements>	See Details ; if nothing is specified, READing is used

Details

This query makes the number of readings specified by [:SENSe[1]] :COUNT. If multiple readings are made, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command. To get multiple readings, use the :TRACe:DATA? command.

NOTE

To change the number of digits returned in a remote command reading, use the :FORMat:ASCIi:PRECision command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.
- Use a comma to delineate multiple elements for a data point.

The options for <bufferElements> are described in the following table.

Option	Description
CHANnel	The channel from which the data was acquired
DATE	The date when the data point was measured; the buffer style must be set to the style standard or full to use this option
EXTRa	Returns an additional value (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAFORMatted	Returns the measurement and the unit of measure of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAUNIT	Returns the units of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
FORMatted	The measured value as it appears on the front panel
FRACTIONal	The fractional seconds when the data point was measured
READING	The measurement reading
RELative	The relative time when the data point was measured
SECONDS	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
STATUS	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time when the data point was measured
TSTamp	The timestamp when the data point was measured
UNIT	The unit of measure of the measurement

The output of :READ? is affected by the data format selected by :FORMat[:DATA]. If you set FORMat[:DATA] to REAL or SREAL, you will have fewer options for buffer elements. The only buffer elements available are READING, RELative, and EXTRa. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

The STATUS buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high

0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high
0x0100	STAT_START_GROUP	256	First reading in a group

Example

```
:TRACe:MAKE "voltMeasBuffer", 10000
:SENSe:FUNCTion "VOLTage"
:COUN 10
:READ? "voltMeasBuffer", FORM, DATE, READ
:TRAC:DATA? 1, 10, "voltMeasBuffer"
```

Create a buffer named `voltMeasBuffer`.

Set the measurement function to voltage.

Set the count to 10.

Make the measurements and store them in the buffer `voltMeasBuffer`. Return the last reading as displayed on the front panel with the date, along with the unformatted reading.

Return all 10 readings from the reading buffer.

Example output is:

```
-000.06580 mV,10/14/2018,-6.580474E-05
-1.322940E-05,-7.876178E-05,-7.798489E-05,-7.201674E-05,-9.442933E-05,-7.653603E-
06,-7.916663E-05,-8.177242E-05,-6.187183E-05,-6.580474E-05
```

Also see

[:FETCh?](#) (on page 12-3)

[\[:SENSe\[1\]\]:COUNt](#) (on page 12-185)

[:TRACe:DATA?](#) (on page 12-237)

[:TRACe:TRIGger](#) (on page 12-260)

CALCulate subsystem

The commands in this subsystem configure and control the math and limit operations.

:CALCulate2:<function>:LIMit<Y>:AUDible

This command determines if the instrument beeper sounds when a limit test passes or fails.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Continuity: PASS Other functions: NONE

Usage

```
:CALCulate2:<function>:LIMit<Y>:AUDible <state>
:CALCulate2:<function>:LIMit<Y>:AUDible <state>, (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:AUDible?
:CALCulate2:<function>:LIMit<Y>:AUDible? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<Y>	Limit number: 1 or 2
<state>	When the beeper sounds: ■ Never: NONE ■ On test failure: FAIL ■ On test pass: PASS
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTInuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The tone and length of beeper cannot be adjusted.

Example

<pre>:CALC2:VOLT:LIM1:CLE:AUTO OFF :CALC2:VOLT:LIM1:AUD FAIL :CALC2:VOLT:LIM1:LOW 0.25 :CALC2:VOLT:LIM1:UPP 2.5 :CALC2:VOLT:LIM1:STAT ON :READ? :CALC2:VOLT:LIM1:FAIL? :CALC2:VOLT:LIM1:CLE</pre>	Set limit autoclear off. Enable the beeper for limit 1 when a voltage measurement exceeds the limit. Set lower limit 1 for voltage to 0.25 V. Set upper limit 1 for voltage to 2.5 V. Enable limit 1 testing for voltage. Make a reading; the limit is checked and results display on the front panel. Return the test results; example output if the test fails on the low limit: LOW Clear the test results.

Also see

[:CALCulate2:<function>:LIMit<Y>:STATe](#) (on page 12-26)

:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO

This command indicates if the test result for limit *Y* should be cleared automatically or not.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ON (1)

Usage

```
:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO <state>
:CALCulate2:<function>:CLEar:AUTO <state>, (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO?
:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<Y>	Limit number: 1 or 2
<state>	The auto clear setting: - Disable: OFF or 0 - Enable: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

When autoclear is set to on, limit conditions are cleared automatically after each measurement. If you are making a series of measurements, the instrument shows the limit test result of the last measurement for the pass or fail indication for the limit.

If you want to know if any of a series of measurements failed the limit, set the autoclear setting to off. When this is set to off, a failed indication is not cleared automatically. It remains set until it is cleared with the clear command.

The autoclear setting affects both the high and low limits.

Example

:CALC2:VOLT:LIM1:CLEAR AUTO ON	Set limit autoclear on.
:CALC2:VOLT:LIM1:AUD FAIL	Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
:CALC2:VOLT:LIM1:LOW 0.25	Set lower limit 1 for voltage to 0.25 V.
:CALC2:VOLT:LIM1:UPP 2.5	Set upper limit 1 for voltage to 2.5 V.
:CALC2:VOLT:LIM1:STAT ON	Enable limit 1 testing for voltage.
:READ?	Make a reading; the limit is checked and results display on the front panel.
:CALC2:VOLT:LIM1:FAIL?	Return the test results; example output if the test fails on the low limit: LOW
	The test results are automatically cleared.

Also see

[:CALCulate2:<function>:LIMit<Y>:CLEar\[:IMMediate\]](#) (on page 12-20)

:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate]

This command clears the results of the limit test defined by *Y*.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:CALCulate2:<function>:LIMit<Y>:CLEar[:IMMediate]
:CALCulate2:<function>:CLEar[:IMMediate] (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<Y>	Limit number: 1 or 2
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Use this command to clear the test results of limit *Y* when the limit auto clear option is turned off. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, turn the auto clear option on.

Example

```
:CALC2:VOLT:LIM1:CLE:AUTO OFF
:CALC2:VOLT:LIM1:AUD FAIL
:CALC2:VOLT:LIM1:LOW 0.25
:CALC2:VOLT:LIM1:UPP 2.5
:CALC2:VOLT:LIM1:STAT ON
:READ?
:CALC2:VOLT:LIM1:FAIL?
:CALC2:VOLT:LIM1:CLE
```

Set limit autoclear off.

Enable the beeper for limit 1 when a voltage measurement exceeds the limit.

Set lower limit 1 for voltage to 0.25 V.

Set upper limit 1 for voltage to 2.5 V.

Enable limit 1 testing for voltage.

Make a reading; the limit is checked and results display on the front panel.

Return the test results; example output if the test fails on the low limit:

LOW

Clear the test results.

Also see

[:CALCulate2:<function>:LIMit<Y>:CLEar:AUTO](#) (on page 12-19)

[:CALCulate2:<function>:LIMit<Y>:LOWer\[:DATA\]](#) (on page 12-24)

[:CALCulate2:<function>:LIMit<Y>:UPPer\[:DATA\]](#) (on page 12-27)

:CALCulate2:<function>:LIMIT<Y>:FAIL?

This command queries the results of a limit test.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:CALCulate2:<function>:LIMIT<Y>:FAIL?
:CALCulate2:<function>:LIMIT<Y>:FAIL? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<Y>	Limit number: 1 or 2
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command queries the result of a limit test for the selected measurement function.

The response message indicates if the limit test passed or how it failed (on the high or low limit).

If autoclear is set to off, reading the results of a limit test does not clear the fail indication of the test. To clear a failure, send the clear command. To automatically clear the results, set auto clear on.

If auto clear is set to on and you are making a series of measurements, the last measurement limit determines the fail indication for the limit. If auto clear is turned off, the results return a test fail if any of one of the readings failed.

To use this attribute, you must set the limit state to on.

The results of the limit test for limit *Y*:

- NONE: Test passed; the measurement is between the upper and lower limits
- HIGH: Test failed; the measurement exceeded the upper limit
- LOW: Test failed; the measurement exceeded the lower limit
- BOTH: Test failed; the measurement exceeded both limits

Example

<pre>:CALC2:VOLT:LIM1:CLEAR:AUTO OFF :CALC2:VOLT:LIM1:AUD FAIL :CALC2:VOLT:LIM1:LOW 0.25 :CALC2:VOLT:LIM1:UPP 2.5 :CALC2:VOLT:LIM1:STAT ON :READ? :CALC2:VOLT:LIM1:FAIL? :CALC2:VOLT:LIM1:CLEAR</pre>	Set limit autoclear off. Enable the beeper for limit 1 when a voltage measurement exceeds the limit. Set lower limit 1 for voltage to 0.25 V. Set upper limit 1 for voltage to 2.5 V. Enable limit 1 testing for voltage. Make a reading; the limit is checked and results display on the front panel. Return the test results; example output if the test fails on the low limit: LOW Clear the test results.

Also see

[:CALCulate2:<function>:LIMIT<Y>:CLEAR:AUTO](#) (on page 12-19)

[:CALCulate2:<function>:LIMIT<Y>:CLEAR:IMMEDIATE](#) (on page 12-20)

[:CALCulate2:<function>:LIMIT<Y>:STATE](#) (on page 12-26)

[Limit testing and binning](#) (on page 4-69)

:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]

This command specifies the lower limit for a limit test.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	-1 for most functions; see Details

Usage

```
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] <n>
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] <DEF|MIN|MAX>
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] <n>, (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA] <DEF|MIN|MAX>, (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]?
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]? (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:LOWer[:DATA]? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<Y>	Limit number: 1 or 2
<n>	The low limit value of limit <i>Y</i> : -1E+12 to 1E+12)
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command sets the lower limit for the limit *Y* test for the selected measure function. When limit *Y* testing is enabled, this causes a fail indication to occur when the measurement value is less than this value.

Default is 0.3 for limit 1 when the diode function is selected. The default for limit 2 for the diode function is -1.

Example

```
:CALC2:VOLT:LIM1:CLE:AUTO OFF
:CALC2:VOLT:LIM1:AUD FAIL
:CALC2:VOLT:LIM1:LOW 0.25
:CALC2:VOLT:LIM1:UPP 2.5
:CALC2:VOLT:LIM1:STAT ON
:READ?
:CALC2:VOLT:LIM1:FAIL?
:CALC2:VOLT:LIM1:CLE
```

Set limit autoclear off.
Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
Set lower limit 1 for voltage to 0.25 V.
Set upper limit 1 for voltage to 2.5 V.
Enable limit 1 testing for voltage.
Make a reading; the limit is checked and results display on the front panel.
Return the test results; example output if the test fails on the low limit:
LOW
Clear the test results.

Also see

[:CALCulate2:<function>:LIMit<Y>:UPPer\[:DATA\]](#) (on page 12-27)

:CALCulate2:<function>:LIMit<Y>:STATe

This command enables or disables a limit test on the measurement from the selected measure function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
:CALCulate2:<function>:LIMit<Y>:STATe <state>
:CALCulate2:<function>:LIMit<Y>:STATe <state>, (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:STATe?
:CALCulate2:<function>:LIMit<Y>:STATe? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<Y>	Limit number: 1 or 2
<state>	Disable the limit test: OFF or 0 Enable the limit test: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command enables or disables a limit test for the selected measurement function. When this attribute is enabled, the limit *Y* testing occurs on each measurement made by the instrument. Limit *Y* testing compares the measurements to the high-limit and low-limit values. If a measurement falls outside these limits, the test fails.

Example

```
:CALC2:VOLT:LIM1:CLE:AUTO OFF
:CALC2:VOLT:LIM1:AUD FAIL
:CALC2:VOLT:LIM1:LOW 0.25
:CALC2:VOLT:LIM1:UPP 2.5
:CALC2:VOLT:LIMIT1:STAT ON
:READ?
:CALC2:VOLT:LIMIT1:FAIL?
:CALC2:VOLT:LIM1:CLE
```

Set limit autoclear off.
Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
Set lower limit 1 for voltage to 0.25 V.
Set upper limit 1 for voltage to 2.5 V.
Enable limit 1 testing for voltage.
Make a reading; the limit is checked and results display on the front panel.
Return the test results; example output if the test fails on the low limit:
LOW
Clear the test results.

Also see

[:CALCulate2:<function>:LIMit<Y>:CLEAr:AUTO](#) (on page 12-19)
[:CALCulate2:<function>:LIMit<Y>:CLEAr:IMMediate](#) (on page 12-20)
[:CALCulate2:<function>:LIMit<Y>:FAIL?](#) (on page 12-22)
[:CALCulate2:<function>:LIMit<Y>:LOWer\[:DATA\]](#) (on page 12-24)
[:CALCulate2:<function>:LIMit<Y>:UPPer\[:DATA\]](#) (on page 12-27)

:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]

This command specifies the upper limit for a limit test.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1 for most functions; see Details

Usage

```

:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] <value>
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] <DEF|MIN|MAX>
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] <value>, (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA] <DEF|MIN|MAX>, (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]?
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]? <DEF|MIN|MAX>
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]? (@<channelList>)
:CALCulate2:<function>:LIMit<Y>:UPPer[:DATA]? <DEF|MIN|MAX>, (@<channelList>)
  
```

<function>	The function to which the setting applies; see Functions
<Y>	Limit number: 1 or 2
<value>	The value of the upper limit (-1e+12 to 1e+12) or DEFault, MINimum, or MAXimum
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command sets the high limit for the limit *Y* test for the selected measurement function. When limit *Y* testing is enabled, the instrument generates a fail indication when the measurement value is more than this value.

Default is 0.8 for limit 1 when the diode function is selected; 10 when the continuity function is selected. The default for limit 2 for the diode and continuity functions is 1.

Example

```
:CALC2:VOLT:LIM1:CLE:AUTO OFF
:CALC2:VOLT:LIM1:AUD FAIL
:CALC2:VOLT:LIM1:LOW 0.25
:CALC2:VOLT:LIM1:UPP 2.5
:CALC2:VOLT:LIM1:STAT ON
:READ?
:CALC2:VOLT:LIM1:FAIL?
:CALC2:VOLT:LIM1:CLE
```

Set limit autoclear off.
Enable the beeper for limit 1 when a voltage measurement exceeds the limit.
Set lower limit 1 for voltage to 0.25 V.
Set upper limit 1 for voltage to 2.5 V.
Enable limit 1 testing for voltage.
Make a reading; the limit is checked and results display on the front panel.
Return the test results; example output if the test fails on the low limit:
LOW
Clear the test results.

Also see

[:CALCulate2:<function>:LIMit<Y>:LOWer\[:DATA\]](#) (on page 12-24)

[:CALCulate2:<function>:LIMit<Y>:STATe](#) (on page 12-26)

:CALCulate[1]:<function>:MATH:FORMat

This command specifies which math operation is performed on measurements when math operations are enabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	PERC

Usage

```
:CALCulate[1]:<function>:MATH:FORMat <operation>
:CALCulate[1]:<function>:MATH:FORMat <operation>, (@<channelList>)
:CALCulate[1]:<function>:MATH:FORMat?
:CALCulate[1]:<function>:MATH:FORMat? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<operation>	The name of the math operation: ■ y = mx+b: MXB ■ Percent: PERCent ■ Reciprocal: RECiprocal
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This specifies which math operation is performed on measurements for the selected measurement function.

You can choose one of the following math operations:

- **y = mx+b**: Manipulate normal display readings by adjusting the m and b factors.
- **Percent**: Displays measurements as the percentage of deviation from a specified reference constant.
- **Reciprocal**: The reciprocal math operation displays measurement values as reciprocals. The displayed value is $1/X$, where X is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).

Math calculations are applied to the input signal after relative offset and before limit tests.

Example

```
:CALC:VOLT:MATH:FORM MXB
:CALC:VOLT:MATH:MMF 0.80
:CALC:VOLT:MATH:MBF 50
:CALC:VOLT:MATH:STAT ON
```

Set the math function for voltage measurements to $mx+b$.
Set the scale factor for voltage measurements to 0.80.
Set the offset factor to 50.
Enable the math function.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[:CALCulate\[1\]:<function>:MATH:MBFactor](#) (on page 12-31)

[:CALCulate\[1\]:<function>:MATH:MMFactor](#) (on page 12-32)

[:CALCulate\[1\]:<function>:MATH:PERCent](#) (on page 12-34)

[:CALCulate\[1\]:<function>:MATH:STATe](#) (on page 12-36)

:CALCulate[1]:<function>:MATH:MBFactor

This command specifies the offset, *b*, for the $y = mx + b$ operation.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0

Usage

```
:CALCulate[1]:<function>:MATH:MBFactor <n>
:CALCulate[1]:<function>:MATH:MBFactor <DEF|MIN|MAX>
:CALCulate[1]:<function>:MATH:MBFactor <n>, (@<channelList>)
:CALCulate[1]:<function>:MATH:MBFactor <DEF|MIN|MAX>, (@<channelList>)
:CALCulate[1]:<function>:MATH:MBFactor?
:CALCulate[1]:<function>:MATH:MBFactor? <DEF|MIN|MAX>
:CALCulate[1]:<function>:MATH:MBFactor? (@<channelList>)
:CALCulate[1]:<function>:MATH:MBFactor? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The offset for the $y = mx + b$ operation: $-1e12$ to $+1e12$
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURREnt
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This attribute specifies the offset (*b*) for an $mx+b$ operation.

The $mx+b$ math operation lets you manipulate normal display readings (*x*) mathematically based on the calculation:

$$y = mx + b$$

Where:

- *y* is the displayed result
- *m* is a user-defined constant for the scale factor
- *x* is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- *b* is the user-defined constant for the offset factor

Example

```
:CALC:VOLT:MATH:FORM MXB
:CALC:VOLT:MATH:MMF 0.80
:CALC:VOLT:MATH:MBF 50
:CALC:VOLT:MATH:STAT ON
```

Set the math function for voltage measurements to $mx+b$.
 Set the scale factor for voltage measurements to 0.80.
 Set the offset factor to 50.
 Enable the math function.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[:CALCulate\[1\]:<function>:MATH:FORMat](#) (on page 12-29)

[:CALCulate\[1\]:<function>:MATH:MMFactor](#) (on page 12-32)

[:CALCulate\[1\]:<function>:MATH:STATe](#) (on page 12-36)

:CALCulate[1]:<function>:MATH:MMFactor

This command specifies the scale factor, m , for the $y = mx+b$ math operation.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
:CALCulate[1]:<function>:MATH:MMFactor <value>
:CALCulate[1]:<function>:MATH:MMFactor <DEF|MIN|MAX>
:CALCulate[1]:<function>:MATH:MMFactor <value>, (@<channelList>)
:CALCulate[1]:<function>:MATH:MMFactor <DEF|MIN|MAX>, (@<channelList>)
:CALCulate[1]:<function>:MATH:MMFactor?
:CALCulate[1]:<function>:MATH:MMFactor? <MIN|MAX|DEF>
:CALCulate[1]:<function>:MATH:MMFactor? (@<channelList>)
:CALCulate[1]:<function>:MATH:MMFactor? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions

<value>	The scale factor: $-1e12$ to $+1e12$
<MIN MAX DEF>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command sets the scale factor (m) for an $mx+b$ operation for the selected measurement function.

The $mx+b$ math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$$y = mx + b$$

Where:

- y is the displayed result
- m is a user-defined constant for the scale factor
- x is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- b is the user-defined constant for the offset factor

Example

<pre>:CALC:VOLT:MATH:FORM MXB :CALC:VOLT:MATH:MMF 0.80 :CALC:VOLT:MATH:MBF 50 :CALC:VOLT:MATH:STAT ON</pre>	Set the math function for voltage measurements to $mx+b$. Set the scale factor for voltage measurements to 0.80. Set the offset factor to 50. Enable the math function.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)
[:CALCulate\[1\]:<function>:MATH:FORMat](#) (on page 12-29)
[:CALCulate\[1\]:<function>:MATH:MBFactor](#) (on page 12-31)
[:CALCulate\[1\]:<function>:MATH:STATe](#) (on page 12-36)

:CALCulate[1]:<function>:MATH:PERCent

This command specifies the reference constant that is used when math operations are set to percent.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
:CALCulate[1]:<function>:MATH:PERCent <value>
:CALCulate[1]:<function>:MATH:PERCent <DEF|MIN|MAX>
:CALCulate[1]:<function>:MATH:PERCent <value>, (@<channelList>)
:CALCulate[1]:<function>:MATH:PERCent <DEF|MIN|MAX>, (@<channelList>)
:CALCulate[1]:<function>:MATH:PERCent?
:CALCulate[1]:<function>:MATH:PERCent? <DEF|MIN|MAX>
:CALCulate[1]:<function>:MATH:PERCent? (@<channelList>)
:CALCulate[1]:<function>:MATH:PERCent? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<value>	The reference used when the math operation is set to percent: -1e12 to +1e12
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This is the constant that is used when the math operation is set to percent.

The percent math function displays measurements as percent deviation from a specified reference constant. The percent calculation is:

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

Where:

- *Percent* is the result
- *Input* is the measurement (if relative offset is being used, this is the relative offset value)
- *Reference* is the user-specified constant

Example

```
CALC:VOLT:MATH:FORM PERC
CALC:VOLT:MATH:PERC 50
CALC:VOLT:MATH:STAT ON
```

```
Set the math operations for voltage to percent.
Set the percentage value to 50.
Enable math operations.
```

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[:CALCulate\[1\]:<function>:MATH:FORMat](#) (on page 12-29)

[:CALCulate\[1\]:<function>:MATH:STATe](#) (on page 12-36)

:CALCulate[1]:<function>:MATH:STATe

This command enables or disables math operation.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
:CALCulate[1]:<function>:MATH:STATe <n>
:CALCulate[1]:<function>:MATH:STATe <n>, (@<channelList>)
:CALCulate[1]:<function>:MATH:STATe?
:CALCulate[1]:<function>:MATH:STATe? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	Disable math operations: OFF or 0 Enable math operations: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

When this command is set to on, the math operation specified by the math format command is performed before completing a measurement.

Example

```
:CALC:VOLT:MATH:FORM MXB
:CALC:VOLT:MATH:MMF 0.80
:CALC:VOLT:MATH:MBF 50
:CALC:VOLT:MATH:STAT ON
```

Set the math function for voltage measurements to mx+b.
Set the scale factor for voltage measurements to 0.80.
Set the offset factor to 50.
Enable the math function.

Also see

[:CALCulate\[1\]:<function>:MATH:FORMat](#) (on page 12-29)
[Calculations that you can apply to measurements](#) (on page 4-59)

CALibration subsystem

These calibration commands are used with the 7706, 7708, and 7710 modules.

The commands in this subsystem are protected by the calibration lock except for queries and :CALibration:PROTected:CODE.

:CALibration:PROTected:CARD1:COUNT?

NOTE

This command is only for use with the 7706, 7708, and 7710 modules.

This command returns the number of times a module has been calibrated.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:CALibration:PROTected:CARD1:COUNT?

Details

You can use this command to monitor how many times the card in slot 1 has been calibrated.

Example

:CAL:PROT:CARD1:COUN?	Request the calibration count for the module in slot 1.

Also see

None

:CALibration:PROTected:CARD1:DAC:STEP0

NOTE

This command is only for use with the 7706 modules.

This command runs analog output (DAC) calibration.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:CALibration:PROTected:CARD1:DAC:STEP0
```

Details

Refer to the module instructions for detail on making connections and running calibration.

Example

<pre>:CAL:PROT:CODE "KI006510" :CAL:PROT:CARD1:DAC:STEP0 :CAL:PROT:CARD1:SAVE :CAL:PROT:CARD1:LOCK</pre>	Unlock calibration. Perform 7706 analog output calibration. Save calibration. Lock calibration.

Also see

None

:CALibration:PROTected:CARD1:DATE?

NOTE

This command is only for use with the 7706, 7708, and 7710 modules.

This command returns the last calibration date from a module.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:CALibration:PROTected:CARD1:DATE?
```

Details

The module calibration date is automatically set to the date on the DAQ6510 when the module is calibrated.

Example

<pre>:CAL:PROT:CARD1:DATE?</pre>	Request the calibration date for the module in slot 1.

Also see

None

:CALibration:PROTected:CARD1:LOCK

NOTE

This command is only for use with the 7706, 7708, and 7710 modules.

This command locks out module calibration.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

:CALibration:PROTected:CARD1:LOCK
:CALibration:PROTected:CARD1:LOCK?

Details

This command allows you to lock out calibration after completing the procedure.

If you are querying, the return can be:

- 0: Calibration locked.
- 1: Calibration unlocked.

To unlock calibration, use :CALibration:PROTected:CODE.

Example

:CAL:PROT:CARD1:LOCK	Lock out calibration for the module in slot 1.
:CAL:PROT:CARD1:LOCK?	Request the state of the calibration for the module in slot 1.

Also see

[:CALibration:PROTected:CODE](#) (on page 12-42)

:CALibration:PROTected:CARD1:RCOunt

This command resets the card calibration count.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Nonvolatile memory	Not applicable

Usage

:CALibration:PROTected:CARD1:RCOunt

Details

Resets the card calibration count that is reported by :CALibration:PROTected:CARD1:COUNT? to 0.

Example

:CAL:PROT:CARD1:RCO	Reset the card calibration count.

Also see

[:CALibration:PROTected:CARD1:COUNT?](#) (on page 12-37)

:CALibration:PROTected:CARD1:REFTemp?

This command returns the card ground voltage and calibration temperature sensor readings.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:CALibration:PROTected:CARD1:REFTemp?

Details

This query allows you to read the ground voltage and calibration reference temperatures from a module. The response values are delimited by a comma.

Example

:CAL:PROT:CARD1:REFT?
Request card temperature. Example output is: 7.7043E-05,25.2925,25.1029,25.1029,24.9813,25.1933,25.0611,25.0611,24.9282 Where 7.7043E-05 is the ground reference value and the following eight values are temperature sensor readings.

Also see

None

:CALibration:PROTected:CARD1:SAVE

NOTE

This command is only for use with the 7706, 7708, and 7710 modules.

This command saves the calibration constants to the internal memory of the module.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:CAL:PROT:CARD1:SAVE

Details

This command stores the calibration constants that were calculated during module calibration. The constants are stored in nonvolatile memory on the module.

Calibration constants are stored indefinitely once saved.

Example

```
:CAL:PROT:CARD1:SAVE
```

Save module calibration constants.

Also see

[:CALibration:PROTected:CARD1:LOCK](#) (on page 12-39)

:CALibration:PROTected:CARD1:STEP0

NOTE

This command is only for use with the 7706, 7708, and 7710 modules.

This command runs temperature calibration for the module installed in slot 1.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Calibration	Not applicable

Usage

:CALibration:PROTected:CARD1:STEP0 <TEMP>

<TEMP>	Cold calibration temperature (0 °C to 50 °C)

Details

This command calibrates the temperature sensor of the module. The module must be allowed to cool down to ambient temperature before calibration, and the cold temperature of the module must be measured and sent as the <TEMP> parameter during calibration.

NOTE

Before calibrating the temperature sensor on the 7706, make sure that power has been removed from the module for at least two hours to allow module circuitry to cool down. After turning on the power during the calibration procedure, complete the procedure as quickly as possible to minimize module heating that could affect calibration accuracy.

Example

:CAL:PROT:CARD1:STEP0 23	Perform temperature calibration with module at a temperature of 23 °C (measured with an external probe).

Also see

None

:CALibration:PROTected:CODE

NOTE

This command is only for use with the 7706, 7708, and 7710 modules.

This command sends the calibration password so that you can perform calibration procedures.

Type	Affected by	Where saved	Default value
Command only	None	Not saved	KI006510

Usage

:CALibration:PROTected:CODE "<Password>"

<Password>	The calibration password, KI006510

Details

This command enables the DAQ6510 calibration commands. This command must be sent to the instrument before sending any other calibration commands.

Only send :CALibration:PROTected:CODE once before calibrating. You do not need to send it before each calibration step.

Example

:CAL:PROT:CODE "KI006510"	Unlock calibration.

Also see

[:CALibration:PROTected:CARD1:LOCK](#) (on page 12-39)

DIGital subsystem

The commands in the DIGital subsystem control the digital I/O lines.

NOTE

The commands in this subsystem require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

:DIGital:LINE<n>:MODE

This command sets the mode of the digital I/O line to be a digital line, trigger line, or synchronous line and sets the line to be input, output, or open-drain.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	DIG, IN

Usage

```
:DIGital:LINE<n>:MODE <lineType>, <lineDirection>  
:DIGital:LINE<n>:MODE?
```

<n>	The digital I/O line: 1 to 6
<lineType>	Sets the digital line control type; the options are: - Allow direct digital control of the line: DIGital - Configure for trigger control: TRIGger - Configure as a synchronous master or acceptor: SYNchronous
<lineDirection>	Sets the line direction; the options are: - Input: IN - Output: OUT - Open drain: OPENdrain - Master: MASTer - Acceptor: ACceptor See Details for valid combinations with line type.

Details

You can specify the line type and line direction parameters to configure each digital I/O line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or synchronous acceptor

A digital line allows direct control of the digital I/O lines by writing a bit pattern to the lines. A trigger line uses the digital I/O lines to detect triggers.

Set `<lineDirection>` to one of the values shown in the following table.

Value	Description
IN	If the type is digital control, this automatically detects externally generated logic levels. You can read an input line, but you cannot write to it. If the type is trigger control, the line automatically responds to and detects externally generated triggers. It detects falling-edge, rising-edge, or either-edge triggers as input. This mode uses the edge setting specified by <code>:TRIGger:DIGital<n>:IN:EDGE</code>.
OUT	If the type is digital control, you can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low. If the type is trigger control, it is automatically set high or low depending on the output logic setting. Use the negative logic setting when you want to generate a falling edge trigger and use the positive logic setting when you want to generate a rising edge trigger.
OPENdrain	Configures the line to be an open-drain signal. This makes the line compatible with other instruments that use open-drain digital I/O lines or trigger signals, such as other Keithley products. If the type is digital control, the line can serve as an input, an output, or both. You can read from the line or write to it. When a digital I/O line is used as an input in open-drain mode, you must write a 1 to it. If the type is trigger control, you can use the line to detect input triggers or generate output triggers. This mode uses the edge setting specified by <code>:TRIGger:DIGital<n>:IN:EDGE</code>.
ACceptor	Only available with the <code>SYNChronous</code> trigger type. This value detects a falling-edge trigger as an input trigger and automatically latches and drives the trigger line low. Asserting the output trigger releases the latched line.
MASTer	Only available with the <code>SYNChronous</code> trigger type. This value detects a rising-edge trigger as an input. It asserts a TTL-low pulse for output.

Example

<code>:DIG:LINE1:MODE DIG, OUT</code>	Set digital I/O line 1 as a digital output line.

Also see

[Digital I/O lines](#) (on page 8-5)

[Digital I/O port configuration](#) (on page 8-3)

[:TRIGger:DIGital<n>:IN:EDGE](#) (on page 12-303)

:DIGital:LINE<n>:STATe

This command sets a digital I/O line high or low when the line is set for digital control and returns the state on the digital I/O lines.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	See Details

Usage

```
:DIGital:LINE<n>:STATe <state>
:DIGital:LINE<n>:STATe?
```

<n>	The digital I/O line: 1 to 6
<state>	Clear the bit (bit low): 0 Set the bit (bit high): 1

Details

When the line mode for a digital I/O line is set to digital output (:DIG:LINE<n>:MODE DIG, OUT), you can set the line high or low using the <state> parameter. When the line mode is set to digital input (:DIG:LINE<n>:MODE DIG, IN), you can query the state of the digital input line.

When a reset occurs, the digital line state can be read as high because the digital line is reset to a digital input. A digital input floats high if nothing is connected to the digital line.

This returns the integer equivalent values of the binary states on all six digital I/O lines.

Set the state to zero (0) to clear the bit; set the state to one (1) to set the bit.

Example 1

:DIG:LINE1:MODE DIG, OUT :DIG:LINE1:STAT 1	Set digital I/O line 1 as a digital output line. Sets line 1 (bit B1) of the digital I/O port high.

Example 2

:DIG:LINE1:MODE DIG, IN :DIG:LINE1:STAT?	Set digital I/O line 1 as a digital input line. Query the state of line 1 on the digital I/O port. Output: 1

Also see

- [Digital I/O port configuration](#) (on page 8-3)
- [:DIGital:LINE<n>:MODE](#) (on page 12-44)
- [:DIGital:READ?](#) (on page 12-47)
- [:DIGital:WRITe <n>](#) (on page 12-48)
- [:TRIGger:DIGital<n>:IN:EDGE](#) (on page 12-303)

:DIGital:READ?

This command reads the digital I/O port.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:DIGital:READ?

Details

The binary equivalent of the returned value indicates the value of the input lines on the digital I/O port. The least significant bit (bit B1) of the binary number corresponds to digital I/O line 1; bit B6 corresponds to digital I/O line 6.

For example, a returned value of 42 has a binary equivalent of 101010, which indicates that lines 2, 4, 6 are high (1), and the other lines are low (0).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

Example

:DIG:READ?	Assume lines 2, 4, and 6 are set high when the I/O port is read. Output: 42 This is binary 101010

Also see

- [Digital I/O bit weighting](#) (on page 8-13)
- [Digital I/O port configuration](#) (on page 8-3)

:DIGital:WRITe <n>

This command writes to all digital I/O lines.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:DIGital:WRITe <n>
```

<n>

The value to write to the port (0 to 63)

Details

This function writes to the digital I/O port by setting the binary state of each digital line from an integer equivalent value.

The binary representation of the value indicates the output pattern to be written to the I/O port. For example, a value of 63 has a binary equivalent of 111111 (all lines are set high); a *data* value of 42 has a binary equivalent of 101010 (lines 2, 4, and 6 are set high, and the other three lines are set low).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

Example

:DIG:WRIT 63

Sets digital I/O lines 1 through 6 high (binary 111111).

Also see

[Digital I/O bit weighting](#) (on page 8-13)

[Digital I/O port configuration](#) (on page 8-3)

DISPlay subsystem

This subsystem contains commands that control the front-panel display.

:DISPlay:BUFFer:ACTive

This command determines which buffer is used for measurements that are displayed on the front panel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1

Usage

```
:DISPlay:BUFFer:ACTive "<bufferName>"
:DISPlay:BUFFer:ACTive?
```

<bufferName>	The name of the buffer to make active

Details

The buffer defined by this command is used to store measurements data and is shown in the reading buffer indicator on the home screen of the instrument.

Example

:DISP:BUFF:ACT "buffer2"	Set the front panel to use <code>buffer2</code> as the active reading buffer.

Also see

None

:DISPlay:CLEar

This command clears the text from the front-panel USER swipe screen.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:DISPlay:CLEar
```

Example

<pre>DISP:CLE DISP:SCR SWIPE_USER DISP:USER1:TEXT "Batch A122" DISP:USER2:TEXT "Test running"</pre>	Clear the USER swipe screen. Display the USER swipe screen. Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

[:DISPlay:USER<n>:TEXT\[:DATA\]](#) (on page 12-55)

:DISPlay:<function>:DIGits

This command determines the number of digits that are displayed for measurements on the front panel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	See Functions and defaults

Usage

```
:DISPlay:<function>:DIGits <value>
:DISPlay:<function>:DIGits <DEF|MIN|MAX>
:DISPlay:<function>:DIGits <value>, (@<channelList>)
:DISPlay:<function>:DIGits <DEF|MIN|MAX>, (@<channelList>)
:DISPlay:<function>:DIGits?
:DISPlay:<function>:DIGits? <DEF|MIN|MAX>
:DISPlay:<function>:DIGits? (@<channelList>)
:DISPlay:<function>:DIGits? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<value>	Display digits: 6.5: 6 5.5: 5 4.5: 4 3.5: 3
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions and defaults

Function	Def	Function	Def	Function	Def	Function	Def
VOLTage[:DC]	6	TEMPerature	3	RESistance	6	VOLTage[:DC]:RATio	6
VOLTage:AC	6	CONTinuity	4	FRESistance	6	DIGitize:VOLTage	4
CURRent[:DC]	6	FREQuency[:VOLTage]	6	DIODE	6	DIGitize:CURRent	4
CURRent:AC	6	PERiod[:VOLTage]	6	CAPacitance	4		

Details

This command affects how the reading for a measurement is displayed on the front panel of the instrument. It does not affect the number of digits returned in a remote command reading. It also does not affect the accuracy or speed of measurements.

The display digits setting is saved with the function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

The change in digits occurs the next time a measurement is made.

To change the number of digits returned in a remote command reading, use
:FORMat:ASCIi:PRECision.

Example

:DISP:CURR:DIG 5	Set the front panel to display current measurements with 5½ digits.

Also see

[:FORMat:ASCIi:PRECision](#) (on page 12-57)

:DISPlay:LIGHt:STATe

This command sets the light output level of the front-panel display.

Type	Affected by	Where saved	Default value
Command and query	Power cycle	Not applicable	ON50

Usage

:DISPlay:LIGHt:STATe <brightness> :DISPlay:LIGHt:STATe?	
<brightness>	The brightness of the display: ■ Full brightness: ON100 ■ 75% brightness: ON75 ■ 50% brightness: ON50 ■ 25% brightness: ON25 ■ Display off: OFF ■ Display and all indicators off: BLACkout

Details

This command changes the light output of the front panel when a test requires different instrument illumination levels.

The change in illumination is temporary. The normal backlight settings are restored after a power cycle. You can use this to reset a display that is already dimmed by the front-panel Backlight Dimmer.

NOTE

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

Example

DISP:LIGH:STAT ON50	Set the display brightness to 50%.

Also see

[Adjust the backlight brightness and dimmer](#) (on page 3-6)

:DISPlay:READIng:FORMat

This command determines the format that is used to display measurement readings on the front-panel display of the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	PREF

Usage

:DISPlay:READIng:FORMat <format>
:DISPlay:READIng:FORMat?

<format>	Use exponent format: EXPonent Add a prefix to the units symbol, such as k, m, or μ : PREFix

Details

This setting persists through *RST and power cycles.

When Prefix is selected, prefixes are added to the units symbol, such as k (kilo) or m (milli). When Exponent is selected, exponents are used instead of prefixes. When the prefix option is selected, very large or very small numbers may be displayed with exponents.

Example

DISP:READ:FORM EXP	Change front-panel display to show readings in exponential format.

Also see

[Setting the display format](#) (on page 3-76)

:DISPlay:SCReen

This command changes which front-panel screen is displayed.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:DISPlay:SCReen <screenName>

<screenName>	The screen to display: ■ Home screen: HOME ■ Home screen with large readings: HOME_LARGE_reading ■ Reading table: READING_table ■ Graph screen (opens last selected tab): GRAPH ■ Histogram screen: HISTogram ■ FUNCTIONS swipe screen: SWIPE_FUNCTIONS ■ GRAPH swipe screen: SWIPE_GRAPH ■ SECONDARY swipe screen: SWIPE_SECondary ■ SETTINGS swipe screen: SWIPE_SETTings ■ STATISTICS swipe screen: SWIPE_STATistics ■ USER swipe screen: SWIPE_USER (only displays USER swipe screen if user text is sent) ■ CHANNEL swipe screen: SWIPE_CHANnel ■ NONSWITCH swipe screen: SWIPE_NONSwitch ■ SCAN swipe screen: SWIPE_SCAN ■ Channel control screen: CHANNEL_CONTrol ■ Channel settings screen: CHANNEL_SETTings ■ Channel scan screen: CHANNEL_SCAN ■ Open a screen that uses minimal CPU resources: PROCessing

Details

The scan and channel options are only available if you have a card installed and if the front-panel TERMINALS switch is set to REAR. The nonswitch option is only available for cards such as the 7706 and 7707 that have nonswitching channels.

Example

```
DISP:CLE
DISP:USER1:TEXT "Batch A122"
DISP:USER2:TEXT "Test running"
DISP:SCR SWIPE_USER
```

Clear the USER swipe screen.
Set the first line of the USER swipe screen to read "Batch A122" and the second line to display "Test running".
Display the USER swipe screen.



Also see

None

:DISPlay:USER<n>:TEXT[:DATA]

This command defines the text that is displayed on the front-panel USER swipe screen.

Type	Affected by	Where saved	Default value
Command only	Power cycle Instrument reset	Not applicable	Not applicable

Usage

:DISPlay:USER<n>:TEXT[:DATA] "<textMessage>"

<n>	The line of the USER swipe screen on which to display text: - Top line: 1 - Bottom line: 2
<textMessage>	String that contains the message; up to 20 characters for USER1 and 32 characters for USER2

Details

This command defines text messages for the USER swipe screen.

If you enter too many characters, the instrument displays a warning event and shortens the message to fit.

When the instrument is reset, the user test is removed and the USER swipe screen is hidden until another message is defined.

Example

DISP:CLE DISP:SCR SWIPE_USER DISP:USER1:TEXT "Batch A122" DISP:USER2:TEXT "Test running"	Clear the USER swipe screen Display the USER swipe screen. Set the first line to read "Batch A122" and the second line to display "Test running".

Also see

[:DISPlay:SCReen](#) (on page 12-53)

:DISPlay:WATCh:CHANnels

This command determines which channels are set to be watch channels on the front panel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	None

Usage

:DISPlay:WATCh:CHANnels (@<channelList>)
:DISPlay:WATCh:CHANnels?

<channelList>	The channels to set, using standard channel naming (on page 11-6)

Details

Watch channels are channels that you want to focus attention on. Watch channels affect what you see on the CHANNEL and STATISTICS swipe screens. They also determine which readings you see on the home screen.

In the Reading Table, you can select the watch channels to filter the buffer so that only information from the watched channels is shown.

In the Graph screens, you can select the watch channels. Each channel in the watch channels list is displayed as a trace on the graph.

You can define up to 20 channels as watch channels.

Example

DISP:WATC:CHAN (@102:105)	Sets the instrument to watch channels 2, 3, 4, and 5.

Also see

None

FORMat subsystem

The commands for this subsystem select the data format that is used to transfer instrument readings over the remote interface.

:FORMat:ASCIi:PRECision

This command sets the precision (number of digits) for all numbers returned in the ASCII format.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:FORMat:ASCIi:PRECision <value>  
:FORMat:ASCIi:PRECision <DEF|MIN|MAX>  
:FORMat:ASCIi:PRECision?  
:FORMat:ASCIi:PRECision? <DEF|MIN|MAX>
```

<value>	The precision: - Automatic: 0 - Specific value: 1 to 16
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value

Details

This attribute specifies the precision (number of digits) for queries.

Note that the precision is the number of significant digits. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

```
:FORM:ASC:PREC 10
```

Set a precision of 10 digits. An example of the output is:
-6.999999881E-01

Also see

[:FORMat\[:DATA\]](#) (on page 12-58)

:FORMat:BORDER

This command sets the byte order for the IEEE Std 754 binary formats.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	SWAP

Usage

```
:FORMat:BORDER <name>  
:FORMat:BORDER?
```

<name>	The binary byte order: - Normal byte order: NORMal - Reverse byte order for binary formats: SWAPped

Details

This attribute selects the byte order in which data is written.

The `SWAPped` byte order must be used when transmitting binary data to a computer with a Microsoft Windows operating system.

The ASCII data format can only be sent in the normal byte order. If the ASCII format is selected, the `SWAPped` selection is ignored.

When you select `NORMal` byte order, the data format for each element is sent as follows:

Byte 1 Byte 2 Byte 3 Byte 4

(Single precision)

When you select `SWAPped`, the data format for each element is sent as follows:

Byte 4 Byte 3 Byte 2 Byte 1

(Single precision)

The `#0` header is not affected by this command. The header is always sent at the beginning of the data string for each measurement conversion.

Example

FORM:BORD NORM	Use the normal byte order.

Also see

[:FORMat\[:DATA\]](#) (on page 12-58)

:FORMat[:DATA]

This command selects the data format that is used when transferring readings over the remote interface.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	ASC

Usage

:FORMat[:DATA] <type>
:FORMat[:DATA] ?

<type>	The data format, which can be one of the following: ■ ASCII format: <code>ASCIi</code> ■ IEEE Std. 754 double-precision format: <code>REAL</code> ■ IEEE Std. 754 single-precision format: <code>SREal</code>

Details

This command affects the output of `READ?`, `FETCh?`, `MEASure:<function>?`, and `TRACe:DATA?` queries over a remote interface. All other queries are returned in the ASCII format.

NOTE

The DAQ6510 only responds to input commands using the ASCII format, regardless of the data format that is selected for output strings.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with #0 and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

If you set this to `REAL` or `SREAL`, you have fewer options for buffer elements with the `TRACe:DATA?`, `READ?`, `MEASURE:<function>?`, and `FETCh?` commands. The only buffer elements available are `READing`, `RElative`, and `EXTRa`. If you request a buffer element that is not available, you see the event code 1133, "Parameter 4, Syntax error, expected valid name parameter."

Example

```
FORM REAL
```

Set the format to double-precision format.

Also see

[:TRACe:DATA?](#) (on page 12-237)

ROUTe subsystem

The ROUTe subsystem contains commands to open, close, and set up scans for channels. It also contains a command you can use to verify whether the front or rear terminals are used for measurements.

:ABORt

This command stops all trigger model commands and scans on the instrument.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:ABORt

Details

When this command is received, the instrument stops the trigger model and scans.

Also see

[Aborting the trigger model](#) (on page 8-67)

[Trigger model](#) (on page 8-35)

:INITiate[:IMMediate]

This command starts the trigger model or scan.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:INITiate[:IMMediate]

Example

```
INIT
*WAI
```

Starts the trigger model or scan and then waits until the commands are complete to accept new commands.

Also see

[:ABORt](#) (on page 12-60)

[:TRIGger:PAUSe](#) (on page 12-333)

[:TRIGger:RESume](#) (on page 12-333)

[Trigger model](#) (on page 8-35)

:ROUTe[:CHANnel]:CLOSe

This command closes the channels and channel pairs that are specified by the channel list parameter.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
:ROUTe[:CHANnel]:CLOSe (@<channelList>)
:ROUTe[:CHANnel]:CLOSe?
```

<channelList>	A list of the channels to close, using standard channel naming (on page 11-6)

Details

The action of the close command depends on which, if any, function is set for the DMM.

If no function is set, the listed channels or channel pairs are closed. You can select multiple channels.

If the DMM for the channel is set to a function, the listed channels or channel pairs are closed. In addition, it opens channels or channel pairs that could affect the measurements. When a channel is set to a function, only one channel can be specified in the channel list.

When you close a channel or channel pair, the instrument:

- Closes the items in the list of channels.
- Opens any channels on any slots that interfere with the measurement.
- Incurs the settling time and any user-specified delay.

This command is not available for digital I/O, digital-to-analog converter (DAC analog output), and totalizer channels. If the digital I/O, DAC analog output, or totalizer channel is in a range of channels, the unavailable channel is ignored. If you try to close it individually, an error is generated.

Use the query to return a list of closed measurement channels, including the paired channels for 4-wire measurements. The query does not return nonmeasurement channels. If no channels are closed, (@) is returned.

Example 1

```
ROUT:CLOS (@101)
ROUT:CLOS?
ROUT:CLOS (@102)
ROUT:CLOS?
```

Assuming a switching module with at least 10 channels is installed with no functions set, this example closes channels 101, then closes 102 without opening 101.

After closing channel 101, the return for the queried channel is:
(@101)

After closing 102, the return is:
(@101, 102)

Example 2

```
SENS:FUNC 'VOLT', (@101)
SENS:VOLT:NPLC 1, (@101)
SENS:FUNC 'RES', (@102)
SENS:VOLT:NPLC 1, (@102)
ROUT:CLOS (@101)
READ?
ROUT:CLOS?
ROUT:CLOS (@102)
READ?
ROUT:CLOS?
```

This example sets channel 1 on slot 1 to measure voltage with an NPLC of 1.

After closing 101, a DC measurement is made and the return for the queried channel is:

(@101)

After closing 102, a 2-wire resistance measurement is made, and the returned value for the queried channel is:

(@102)

Also see

[:ROUTe\[:CHANnel\]:STATe?](#) (on page 12-78)

[:ROUTe:OPEN](#) (on page 12-76)

[:ROUTe\[:CHANnel\]:DELay](#) (on page 12-68)

:ROUTe[:CHANnel]:CLOSE:COUNT:INTERval

This command determines how often the instrument stores the channel relay closure count.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	MIN15

Usage

```
:ROUTe[:CHANnel]:CLOSE:COUNT:INTERval <interval>
:ROUTe[:CHANnel]:CLOSE:COUNT:INTERval?
```

<interval>

The count update interval:

- 1 minute: MIN1
- 5 minutes: MIN5
- 10 minutes: MIN10
- 15 minutes: MIN15
- 30 minutes: MIN30
- 60 minutes: MIN60
- 24 hours: HOUR24

Details

This command determines how often channel relay closure counts are stored in nonvolatile memory. If the DAQ6510 is turned off before the count is stored, the counts are lost.

The DAQ6510 stores relay closure counts:

- At the interval set by this command.
- When the counts are queried.

Relay closure counts are updated in temporary memory every time a channel is closed regardless of how it was closed.

The lower the interval, the less chance there is of losing relay counts due to power failures. However, writing to memory more often may reduce scan throughput.

It is good practice to query the count at the end of a program to manually update the count.

The query returns the interval that is presently set for the channel closure count.

Example

```
ROUT:CLOS:COUN:INT MIN30
```

Set the channel count save interval to 30 minutes.

Also see

[:ROUTe\[:CHANnel\]:CLOSE:COUNT?](#) (on page 12-64)

:ROUTe[:CHANnel]:CLOSe:COUNT?

This command returns the number of times the relays have been closed for the specified channels.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:ROUTe[:CHANnel]:CLOSe:COUNT? (@<channelList>)

<channelList>	The channels to set, using standard channel naming (on page 11-6)

Details

The DAQ6510 keeps an internal count of the number of times each relay has been closed. This count can help you determine when relays require replacement. Refer to the switching module documentation for the contact life specifications for the relays.

If channels are specified, the count values are returned in the order in which the channels are specified. If slots are specified, the response lists the channels starting from lowest to highest. Because each slot is processed completely before going to the next, all slot 1 channels are listed before slot 2 channels.

Relay closures are counted only when a relay cycles from open to closed state.

It is good practice to get the relay count at the end of a program. This saves the latest count to memory.

Example

ROUT:CLOS:COUN? (@101,104)	Query the closure count of channels 1 and 4 of a module in slot 1. Example return: 10,3
ROUT:CLOS:COUN? (@allslots)	Query the closure count of channels in all slots. Example return: 10,8,6,3,4,4,2,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1 0,12001,12002,12003,12004,12005,12006,12007 ,12008,12009,12010,12011,12012,12013,12014, 12015,12016,12017,12018,12019,12020,12021,1 2022,12023,12024,12025,12026,12027,12028,12 029,12030,12031,12032,12033,12034,12035,120 36,12037,12038,12039,12040

Also see

[:ROUTe\[:CHANnel\]:CLOSe:COUNT:INTERval](#) (on page 12-63)

:ROUTe[:CHANnel]:CONNect:RULE

This command controls the connection rule for closing and opening channels in the instrument.

Type	Affected by	Where saved	Default value
Command and query	Instrument reset Recall setup	Create configuration script Save setup	CONCurent

Usage

```
:ROUTe[:CHANnel]:CONNect:RULE <BBM|MBB|CONC>
:ROUTe[:CHANnel]:CONNect:RULE?
```

<BBM|MBB|CONC>

- **BBM or 1:** Break before make (BBM) connections for relays in the instrument
- **MBB or 2:** Make before break (MBB) connections for relays in the instrument
- **CONCurent or 0:** Does not guarantee a connection rule. The instrument closes relays as efficiently as possible to improve speed performance without applying a rule

Details

The connection rule describes the order in which switch channels are opened and closed during channel operations. This applies to scanning operation and when using the :ROUTe[:CHANnel]:CLOSe command when a channel is set to a function.

The connection rule affects the operating time of these commands. These commands do not allow the instrument to continue execution until the settle time of the relays has expired.

When the connection rule is set to break before make, the instrument ensures that all switch channels open before any switch channels close. 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 make before break, 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.

With no connection rule (set to CONCurent), 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 closed position. Make sure your device under test can withstand this possible condition. 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.

NOTE

You cannot guarantee the sequence of open and closure operations when the channel connect rule is set to CONCurent. It is highly recommended that you implement cold switching when the channel connect rule is set to CONCurent.

In general, the settling time of single commands that open and close switch channels depends on several 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:

- Break before make: Open settle time + close settle time
- Make before break: Close settle time + open settle time
- CONCurrent: Maximum of open settle time or close settle time

CAUTION

Make before break (also known as hot switching) can dry-weld reed relays so that they will always be on. Hot switching is recommended only when external protection is provided.

Example

```
ROUT:CONN:RULE BBM
```

Sets the channel connect rule in the instrument to BBM (break before make).

Also see

[:ROUTe\[:CHANnel\]:CLOSE](#) (on page 12-61)

:ROUTe[:CHANnel]:CS0hms

This command sets up the specified slot to have a common side path.

NOTE

This command is only applicable for cards that support commonside measurements, such as the Keithley Instruments 7701 module.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	OFF

Usage

```
:ROUTe[:CHANnel]:CSOhms (@<slot>), <state>  
:ROUTe[:CHANnel]:CSOhms? (@<slot>)
```

<slot>	The slot number; can be SLOT1, SLOT2, or ALLSLOTS
<state>	The state of the channels in the slot: ON or OFF

Details

The query commonside setting for the specified slot.

Example

```
ROUTE:OPEN:ALL  
ROUTE:CSOhms (@slot1), ON  
SENSe:FUNCTion "FRESistance", (@101:132)  
ROUTE:CLOSe (@101)  
READ?  
ROUTE:CLOSe (@132)  
READ?  
TRACe:DATA? 1, 32, "defbuffer1"  
ROUTE:OPEN:ALL
```

Open all slots.
Set the channels on slot 1 to have a common side path.
Set slot 1 to measure 4-wire resistance.
Close channel 101.
Make a measurement.
Repeat closing and reading channels for each DUT (channels 102 to 131).

Also see

[:SYSTem:CARD<X>:CSOhms?](#) (on page 12-202)

:ROUTe[:CHANnel]:DELay

This command sets additional delay time for specified channels.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (0 s)

Usage

```
:ROUTe[:CHANnel]:DELay <delay>, (@<channelList>)  
:ROUTe[:CHANnel]:DELay? (@<channelList>)
```

<channelList>	The channels to set, using standard channel naming (on page 11-6)
<delay>	Delay time for the selected channels; minimum is 0 seconds

Details

After a channel closes, a command incurs the delay time indicated in the response for a channel before it completes. However, the internal settling time must elapse before the user delay is incurred. Therefore, the sequence is:

1. Command is processed
2. Channel closes
3. Settling time is incurred
4. Channel delay is incurred
5. Command completes

The channel delay is an additional delay that is added after a channel is closed. You can use this delay to allow additional settling time for a signal on that channel. For most cards, the resolution of the delay is 10 μ s. However, check the documentation for your card to verify. To see if the delay value was modified after setting, query the value.

Setting a delay only applies to switch channels.

The delay being specified may be updated based on the delay resolution of the card.

The delay times are returned in a comma-delimited list in the same order that the channels are specified in the channel list parameter. A value of zero (0) indicates that no additional delay time is incurred before a close command completes.

NOTE

Pseudocards do not support user delays, so this value is always zero (0) if a pseudocard is used.

The query returns the delays for the selected channels.

Example

```
ROUT:DEL 0.1, (@slot1)
ROUT:DEL? (@slot1)
```

Set a delay of 0.1 s for all channels in slot 1.
Query the delay value for that slot. An example return:

```
0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,  
    1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0,0,0,0
```

Also see

None

:ROUTe[:CHANnel]:LABel

This command sets the label associated with a channel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	Empty string

Usage

```
:ROUTE[:CHANnel]:LABEL "<label>", (@<channel>)
:ROUTE[:CHANnel]:LABEL? (@<channel>)
```

<label>	The label to assign to the specified channel; only one channel can be specified
<channel>	The channel for which to set or query the label

Details

This command sets the label of the specified channel to the label value. The label must be unique; you cannot assign the same label to more than one channel. Labels cannot start with a digit. They can be up to 19 characters. On the front panel of the instrument, only the first few characters are displayed.

To clear a label, set it to an empty string ("").

After defining a label, you can use it to specify the channel instead of using the channel number in commands.

The query returns the label associated with the channel. If there is no label set, an empty string is returned.

Example

```
ROUT:LAB "", (@101)
ROUT:LAB "First", (@101)
ROUT:LAB? (@101)
ROUT:LAB? (@First)
```

Remove any existing label from channel 1 on slot 1.
Assign the label name "First" to channel 1.
Query the label using the slot and channel number.
Query the label using the label name.
Both return:
First

Also see

None

:ROUTe[:CHANnel]:MATCH

This command sets the match value on a digital input or totalizer channel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	See Details

Usage

```
:ROUTe[:CHANnel]:MATCH <value>, (@<channelList>)  
:ROUTe[:CHANnel]:MATCH <value>, (@<channelList>), <mask>  
:ROUTe[:CHANnel]:MATCH? (@<channelList>)
```

<value>	Channel value to compare on the specified channel
<channelList>	The channels to set, using standard channel naming (on page 11-6)
<mask>	Value to specify the bits used to mask <value>

Details

You can use matching to set a state or generate an event when the digital I/O or totalizer count matches a value, instead of continually reading the digital input or totalizer.

The default mask is all bits.

Switch channels are not supported. If they are included in a range or slot specifier, they are ignored. If they are specified directly, an error is generated. The query returns the match value on the specified channels.

Example

ROUT:MATC:TYPE EXACT, (@121) ROUT:MATC 32, (@121)	Assuming a 7706 is installed, set the match type for digital I/O channel 21 on slot 1 to EXACT. Set the match value to 32.

Also see

[:ROUTe\[:CHANnel\]:MATCH:TYPE](#) (on page 12-71)

:ROUTe[:CHANnel]:MATCH:TYPE

This command sets the match type on a digital I/O or totalizer channel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:ROUTe[:CHANnel]:MATCH:TYPE <type>, (@<channelList>)  
:ROUTe[:CHANnel]:MATCH:TYPE? (@<channelList>)
```

<type>	The match operation: ■ EXACT: Exact match ■ ANY: The match value or match mask value ■ UNCHanged: Value unchanged ■ NONE: No matching
<channelList>	Channel value to compare on the specified channel, using standard channel naming (on page 11-6)

Details

For an exact match, the state match indicator only becomes true when the match value equals the channel-read value.

For an any match, a match is when the match value equals the channel-read value.

For an unchanged match, the value should be the same as the original. If not, a match is declared.

When none is set, matching is disabled.

This command is not supported on DAC and switch channels.

The query returns the match value that was set for a digital I/O or totalizer channel.

Example

ROUT:MATC:TYPE EXAC, (@121)	Assuming a 7706 is installed, set the match type for digital I/O channel 21 on slot 1 to EXACT.

Also see

[:ROUTe\[:CHANnel\]:MATCH](#) (on page 12-70)

:ROUTe[:CHANnel]:MODE

This command sets the mode of operation of a channel.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	Digital I/O: INPut Totalizer: RISing

Usage

```
:ROUTe[:CHANnel]:MODE <mode>, (@<channelList>)  
:ROUTe[:CHANnel]:MODE? (@<channelList>)
```

<mode>	The mode of operation for a channel For digital channels, the options are: - INPut: Sets the channels as input ports; query returns 8 - OUTPut: Sets the channels as output ports; query returns 9 For totalizer channels, the options are: - RISing: Count on the rising edge of the input signal; query returns 1 - FALLing: Count on the falling edge of the input signal; query returns 0 - RISING_RESET: Count on the rising edge of the input signal; the count is reset when the count is read; query returns 3 - FALLING_RESET: Count on the falling edge of the input signal; the count is reset when the count is read; query returns 2 For DAC channels, the query returns 17.
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Details

You can set digital I/O channels to be input or output channels. Changing the mode from input to output adds an additional channel delay if a channel delay is set.

For totalizer channels, you can set which side of the input signal to count on and whether or not to reset the count when the count is read. To clear the totalizer count at a specific step in a scan, set the mode to either rising reset or falling reset, then use the add scan command to add the totalizer channel.

The specified channel list must define only one channel type. For example, channel list "121:125" is only valid if channels 21, 22, 23, 24, and 25 are the same type. If one of the channels is a different type of channel, the channel list is invalid and an error is generated.

Changing the mode setting can impact the power consumption of the card. The instrument verifies that power is available before changing the mode. If there is not enough power, the command generates an error.

This command is not available for switch and backplane channels.

Example

```
ROUT:MODE FALLING_RESET, (@125)
ROUT:MODE? (@125)
```

Assuming a 7706, set channel 25 in slot 1 to count on the falling edge and reset the count to 0.
Verify the setting. Output is:
2

Also see

[:ROUTE:CHANnel:DELay](#) (on page 12-68)

[:ROUTE:SCAN:ADD](#) (on page 12-82)

:ROUTe[:CHANnel]:MULTiple:CLOSe

This command closes the listed channels without affecting any other channels.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
:ROUTe[:CHANnel]:MULTiple:CLOSe (@<channelList>)
:ROUTe[:CHANnel]:MULTiple:CLOSe?
```

<channelList>	The list of channels to close

Details

This command closes the specified channels without affecting any other channels, including paired channels.

The action of the close command depends on which, if any, function is set for the DMM.

If no function is set, the listed channels or channel pairs are closed. You can select multiple channels.

If the DMM for the channel is set to a function, the listed channels or channel pairs are closed. In addition, it opens channels or channel pairs that could affect the measurements. When a channel is set to a function, only one channel can be specified in the channel list.

When you close a channel or channel pair, the instrument:

- Closes the items in the list of channels.
- Opens any channels on any slots that interfere with the measurement.
- Incurs the settling time and any user-specified delay.

This command is not available for digital I/O, digital-to-analog converter (DAC analog output), and totalizer channels. If the digital I/O, DAC analog output, or totalizer channel is in a range of channels, the unavailable channel is ignored. If you try to close it individually, an error is generated.

The query returns a list of all closed channels, including nonmeasurement channels and paired channels for 4-wire functions.

If the channel list is large, you should use *OPC or *OPC? with the multiple close. Monitor the status model for closure of the operation complete bit.

Example

```
ROUT:MULT:CLOS (@101)
ROUT:MULT:CLOS (@102)
ROUT:MULT:OPEN (@101)
ROUT:OPEN:ALL
ROUT:MULT:CLOS (@101,123)
```

This example opens and closes channels without being set up for making measurement. Closes channel 101. Adds 102 to the closed channels, then opens 101, then opens all channels, then closes 101 and 123 without setup for any measurement.

Also see

[*OPC](#) (on page 15-9)

:ROUTe[:CHANnel]:MULTiple:OPEN

This command opens the channels in the channel list without affecting any others.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:ROUTe[:CHANnel]:MULTiple:OPEN (@<channelList>)

<channelList>	A list of the channels to open

Details

Opens only the specified channels. Backplane relays and paired channels are not affected.

Example

ROUT:MULT:OPEN (@102, 103, 104)	Open channels 102, 103, and 104.

Also see

None

:ROUTe[:CHANnel]:OPEN

This command opens the specified channels and channel pairs.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:ROUTe[:CHANnel]:OPEN (@<channelList>)

<channelList>	A list of the channels to open; ALLSLOTS, SLOT1, and SLOT2 available, which open all channels on all slots, all channels on slot 1, or all channels on slot 2

Details

If the specified channels are not set to a measurement function, this command opens the specified channels without affecting other channels.

If the specified channels are set to a measurement function, their paired channels and backplane channels are also opened.

The settling time associated with a channel must elapse before the command completes. User delay is not added when a relay opens.

You cannot open digital I/O, DAC, and totalizer channels. If you include one of these channels in the channel list, the invalid channel is ignored but other channels open normally.

Example

ROUT:CLOS (@101)	Close channel 101.
ROUT:CLOS (@102)	Close channel 102 without opening 101.
ROUT:OPEN (@ALLSLOTS)	Open all channels.

Also see

[:ROUTe\[:CHANnel\]:OPEN:ALL](#) (on page 12-77)

:ROUTe[:CHANnel]:OPEN:ALL

This command opens all channels on all slots, including nonmeasurement channels.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:ROUTe[:CHANnel]:OPEN:ALL
```

Details

The settling time associated with a channel must elapse before the command completes. User delay is not added when a relay opens.

You cannot open digital I/O, DAC, and totalizer channels. These channels are ignored when you send this command.

Example

ROUT:OPEN:ALL	Opens all channels.

Also see

None

:ROUTe[:CHANnel]:READ?

This command reads a value from a totalizer, DAC, or digital I/O channel.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:ROUTe[:CHANnel]:READ? (@<channelList>)  
:ROUTe[:CHANnel]:READ? (@<channelList>), "<bufferName>"
```

<channelList>	The channels to set, using standard channel naming (on page 11-6)
<bufferName>	The name of the reading buffer where read values are stored; if no buffer is defined, value is stored in defbuffer1

Details

For totalizer channels, if the mode is set to a reset mode, the count is reset when this command is sent.

Example

ROUT:READ? (@125), "defbuffer1"	Assuming a 7706, read the count from the totalizer channel. Store the information in defbuffer1.

Also see

None

:ROUTe[:CHANnel]:STATe?

This command returns the state indicators of the channels in the instrument.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:ROUTe[:CHANnel]:STATe?  
:ROUTe[:CHANnel]:STATe? (@<channelList>)
```

<channelList>	The channels to set, using standard channel naming (on page 11-6); if no channels are defined, all slots are returned

Details

This command returns the overload, match, closed, or open state of a channel. The states that can be returned depend on the type of channel.

All channels can report an open or closed channel.

Totalizer and digital I/O channels can report that a value has been matched.

Totalizer channels can also report that the count has overflowed, which means the last value read was less than the previous value read. This occurs when the totalizer reaches 4,294,967,295 and automatically resets to zero between reads.

Cards are returned sequentially by channel number.

Each bit in the return represents a different indicator. Therefore, multiple indicators can be present (the OR operation is performed bitwise).

Possible returns are:

- 0: Channel is open
- 1: Channel is closed
- 4: Digital I/O or totalizer channel value is matched
- 8: Totalizer channel has overflowed

Example

:ROUT:CLOS (@105) :ROUT:STAT? (@101:120)	Close channel 5 on slot 1. Query the state of the first 20 channels on slot 1. Output (assuming a 7706): 0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Also see

[:ROUTe\[:CHANnel\]:MATCH](#) (on page 12-70)

:ROUTe[:CHANnel]:TYPE?

This command returns the type associated with a channel.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:ROUTe[:CHANnel]:TYPE? (@<channelList>)

<channelList>	List of channels to query

Details

The channel type is defined by the physical hardware of the card on which the channel exists. The following are valid channel types:

- BACK: Backplane channel
- DAC: Digital-analog converter
- DIG: Digital
- DIO: Digital input/output
- POLE: Two-pole or four-pole selection relay
- RAD: Radio frequency
- SWIT: Switch
- TOT: Totalizer

Refer to the documentation for your switching module for information about the channel types available for your switching module.

Example

ROUT:TYPE? (@125)	If a 7706 switching module is installed, channel 125 is a totalizer, so the return is: TOT
ROUT:TYPE? (@101)	If the first channel is a switch, the return is: SWIT

Also see

None

:ROUTe[:CHANnel]:WIDTh

This command allows you to control multiple digital I/O channels as one channel.

Type	Affected by	Where saved	Default value
Command and query	Restore settings Instrument reset Power cycle	Save settings	1

Usage

:ROUTe[:CHANnel]:WIDTh <width>, (@<channelNumber>)
:ROUTe[:CHANnel]:WIDTh? (@<channelNumber>)

<width>	The number of channels to combine
<channelNumber>	The first channel number of the channels to be combined

Details

Changing the width allows for multiple sequential digital I/O channels to be linked. This option only applies to digital I/O channels, such as channels 21 and 22 on the 7706 and channels 11 to 14 on the 7707.

For widths greater than 1, the specified channel occupies the least significant byte. For example, writing the value of 4278255360 (hexadecimal FF00FF00) to channel 1 with a width of 4 sets channel 1 to 0, channel 2 to 255 (hexadecimal FF), channel 3 to 0, and channel 4 to 255 (hexadecimal FF). Writing the value of 4278255360 to channel 1 with a width of 1 sets channel 1 to 0 and leaves other channels untouched.

You must use decimal values when sending commands to the DAQ6510.

The maximum width is the number of digital I/O channels available on the card. If a width crosses channels, then only the channels set to output are affected.

Example

ROUT:WIDTh 2, (@121)	Assuming a 7706 in slot 1, sets the digital I/O channels starting at channel 21 to be handled as one channel.

Also see

None

:ROUTe[:CHANnel]:WRITe

This command writes a value to a channel.

Type	Affected by	Where saved	Default value
Command only	Instrument reset Power cycle	Save settings	Not applicable

Usage

ROUTe[:CHANnel]:WRITe <value>, (@<channelList>)

<value>	The value to be written to the channel (must be a decimal value)
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Details

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.

For DAC channels, the value is expected to be the set 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.

For digital I/O channels, the value becomes the setting of the digital output.

The time it takes to execute the write command is affected by the channel delay setting.

Example

ROUT:WRIT 10, (@121)	Writes 10 to the 7706 digital I/O channels.

Also see

[:ROUTe\[:CHANnel\]:READ?](#) (on page 12-77)

:ROUTe:SCAN:ADD

This command adds channels to the scan list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:ROUTe:SCAN:ADD (@<channelList>)
:ROUTe:SCAN:ADD (@<channelList>), "<configurationList>"
:ROUTe:SCAN:ADD (@<channelList>), "<configurationList>", <index>
```

<channelList>	List of channels to add, in the order in which they should occur in the scan
<configurationList>	A string that defines the configuration list to recall
<index>	The index in the configuration list to recall; default is 1

Details

Use this function to add channels to the present scan list. If the scan list does not exist, it also creates a scan list.

Channels are added to the end of the present list in the order in which they are specified in the channel list.

If you include a configuration list, the configuration list must exist before you send this command.

NOTE

The front-panel SCAN screen does not show settings set by this configuration list parameter. To check settings, use the command `ROUTe:SCAN:CREate?`

Example 1

<pre>SENS:FUNC 'VOLT', (@101:120) SENS:VOLT:NPLC 1, (@101:120) ROUT:SCAN:CRE (@101:110) ROUT:SCAN:ADD (@117, 112, 119) INIT</pre>	This example sets 20 channels to measure DC voltage with an NPLC of 1. It then creates a scan with the first 10 channels. Three of the channels are added to the scan and are scanned in the order 117, 112, and 119. Start the scan.

Example 2

```
*RST
:FUNC "VOLT"
:SENS:VOLT:RANG 10
:VOLT:NPLC 10

:SENS:CONF:LIST:CRE "scanconfig"
:SENS:CONF:LIST:STOR "scanconfig"
:VOLT:NPLC 0.01
:SENS:CONF:LIST:STOR "scanconfig"

:FUNC "VOLT", (@101:104)
:VOLT:NPLC 0.01, (@101:104)

ROUT:SCAN (@101:104)
ROUT:SCAN:ADD (@107, 102, 109), "scanconfig"
ROUT:SCAN:ADD (@107), "scanconfig", 2
```

Set up the DMM for the settings you want to use in the scan. This example shows the function set to DC voltage, with a measurement range of 10 V and the NPLCs set to 10.

Create a configuration list named `scanconfig`.

Store the present configuration to `scanconfig`.

Set NPLC to 0.01.

Store the present configuration to `scanconfig`. This configuration is stored in index 2.

Set up channels 101 to 104 on slot 1 for DC voltage with NPLC set to 0.1.

Create a scan list with a scan list that includes channels 101 to 104.

Add channels 107, 102, and 109 to the end of the scan list, to be scanned in that order. The settings in the configuration list `scanconfig` are used for these channels.

Add channel 107 to the end of the scan list. The settings in index 2 of `scanconfig` are used for this channel.

Also see

[:ROUTe:SCAN\[:CREAt\]e](#) (on page 12-93)

[Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:ADD:SINGLE

This command allows you to include multiple channels in a single scan step.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:ROUTe:SCAN:ADD:SINGLE (@<channelList>)
:ROUTe:SCAN:ADD:SINGLE (@<channelList>), "<configurationList>"
:ROUTe:SCAN:ADD:SINGLE (@<channelList>), "<configurationList>", <index>
```

<channelList>	List of channels to add, in the order in which they should occur in the scan
<configurationList>	A string that defines the configuration list to recall
<index>	The index in the configuration list to recall; default is 1

Details

This command adds a list of channels to be closed simultaneously in a single step of a scan.

If you need to make measurements using multiple functions on these channels, you can use the configuration list parameter to call the function settings. The configuration list must be created before calling it in this command.

NOTE

The front-panel SCAN menu does not show settings set by the configuration list parameter. To check settings, use the command `ROUTe:SCAN:CREATe?`

Example

<pre>SENS:FUNC 'VOLT', (@101:109) SENS:VOLT:NPLC 1, (@101:109) ROUT:SCAN:CRE (@101:109) ROUT:SCAN:ADD:SING (@107, 102, 109) INIT</pre>	This example sets nine channels to measure DC voltage with an NPLC of 1. It then creates a scan with those channels. Three of the channels are added to the scan as a step to be scanned in the order 107, 102, and 109. Start the scan.

Also see

- [:ROUTe:SCAN\[:CREATe\]](#) (on page 12-93)
- [Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:ADD:WRITe

This command writes a specified value to a channel at the added step in the scan.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Nonvolatile memory Save settings	Not applicable

Usage

:ROUTe:SCAN:ADD:WRITe (@<channelList>), <writeValue>

<channelList>	The channels to set, using standard channel naming (on page 11-6)
<writeValue>	The value to write to the channel for this scan step

Details

Allows you to write the DAC or digital I/O in a scan.

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.

Example

:ROUT:SCAN:ADD:WRIT (@124), 2	Assume a 7706 in slot 1. Add a write step to DAC channel 24 to output 2 V.

Also see

[Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:ALARm

This command determines if the scan sends a trigger event when a value is out of limits.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	OFF

Usage

```
:ROUTe:SCAN:ALARm <n>  
:ROUTe:SCAN:ALARm?
```

<n>	Determine whether to generate an alarm notify trigger: - OFF: Do not send notify trigger events when a value is out of limits. - ON: Send a notify trigger event when a value is out of limits.

Details

When this is set on, a trigger is generated when the measurements exceed the limits set for the channels in the scan. To use this trigger, set a stimulus to `SCANALARmlimit`.

Example

<pre>*RST :FUNC "VOLTage", (@101:109) :ROUT:SCAN:CRE (@101:109) :ROUT:SCAN:LEAR:LIM 0.25, 3 :TRIG:EXT:OUT:STIM SCANALARmlimit :ROUT:SCAN:ALAR ON</pre>	Reset the instrument. Set channels 101 to 109 to the DCV measure function. Create a scan that includes channels (101 to 109. Use learn limits to establish the limits to within 25% over three iterations of scan limits. Enable alarm notification on the external trigger out line.

Also see

None

:ROUTe:SCAN:BUFFer

This command defines which buffer is used with the scan.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1

Usage

```
:ROUTe:SCAN:BUFFer "<bufferName>"  
:ROUTe:SCAN:BUFFer?
```

<bufferName>	The reading buffer to use to collect data from the scan

Details

This selects the buffer that stores the data generated by the scan.

Example

ROUT:SCAN:BUFF "defbuffer2"	Sets the buffer for the scan to defbuffer2.

Also see

[Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:BYPass

This command indicates whether the first channel of the scan waits for the channel stimulus event to be satisfied before closing.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	OFF

Usage

:ROUTe:SCAN:BYPass <bypass>
:ROUTe:SCAN:BYPass?

<bypass>	Enable or disable bypass: ■ OFF: Disabled ■ ON: Enabled

Details

When bypass is set to on and the channel stimulus for the scan is set to wait for a stimulus, the first channel of the scan closes when the scan starts (the stimulus setting is ignored).

For other channels, the channel stimulus must be satisfied before the channel action takes place.

When bypass is set to off, every channel (including the first) must satisfy the channel stimulus setting before the channel action occurs for that step.

Example

*RST SENS:FUNC 'RES', (@101:109) ROUT:SCAN:CRE (@101:109) ROUT:SCAN:COUN:SCAN 10 ROUT:SCAN:BYPASS ON ROUT:SCAN:CHAN:STIM DIG1 DIG:LINE1:MODE TRIG, IN TRIG:DIG1:IN:EDGE FALL DIG:LINE3:MODE TRIG, OUT TRIG:DIG3:OUT:LOG NEG TRIG:DIG3:OUT:STIM SCANCHAN INIT	Reset the instrument. Set channels 101 through 109 for 2-wire resistance measurements. Create a scan of channels 101 through 109. Set the scan count to 10. Bypass the trigger for the first channel close. Set the channel close stimulus to respond to a falling edge trigger on digital input line 1. Set a digital output signal to trigger a negative pulse for each time that a defined scan channel is closed. Start the scan.

Also see

[:ROUTe:SCAN:CHANnel:STIMulus](#) (on page 12-89)
[:ROUTe:SCAN:START:STIMulus](#) (on page 12-109)

:ROUTe:SCAN:CHANnel:STIMulus

This command determines which trigger event causes the channel action to occur.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:ROUTe:SCAN:CHANnel:STIMulus <eventID>  
:ROUTe:SCAN:CHANnel:STIMulus?
```

<eventID>	Trigger stimulus used for the channel action; see Details for trigger event IDs

Details

Set the event ID to one of the options in the following table.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMManD
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENDer<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPlete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmlimit (returns NOT3)

Example

```
*RST
SENS:FUNC 'RES', (@101:109)
ROUT:SCAN:CRE (@101:109)
ROUT:SCAN:COUN:SCAN 10
ROUT:SCAN:BYPASS ON
ROUT:SCAN:CHAN:STIM DIG1
DIG:LINE1:MODE TRIG, IN
TRIG:DIG1:IN:EDGE FALL
DIG:LINE3:MODE TRIG, OUT
TRIG:DIG3:OUT:LOG NEG
TRIG:DIG3:OUT:STIM SCANCHAN
INIT
```

Reset the instrument.
Set channels 101 through 109 for 2-wire resistance measurements.
Create a scan of channels 101 through 109.
Set the scan count to 10.
Bypass the trigger for the first channel close.
Set the channel close stimulus to respond to a falling edge trigger on digital input line 1.
Set a digital output signal to trigger a negative pulse for each time that a defined scan channel is closed.
Start the scan.

Also see

None

:ROUTe:SCAN:COUNT:SCAN

This command sets the number of times the scan is repeated.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1

Usage

```
:ROUTe:SCAN:COUNT:SCAN <scanCount>  
:ROUTe:SCAN:COUNT:SCAN?
```

<scanCount>	The scan count value (1 to 100,000,000; set to 0 to set the scan to repeat until aborted)

Details

The scan count attribute setting indicates how many times the scan list is iterated through before the scan completes.

Example

<pre>*RST TRAC:CLE TRACe:POINts 100, "defbuffer1" ROUT:SCAN (@101:109) SENS:FUNC "VOLT", (@101:109) SENS:VOLT:RANG 10, (@101:109) SENS:VOLT:NPLC 0.1, (@101:109) DISP:VOLT:DIG 5, (@101:109) ROUT:SCAN:CRE (@101:109) ROUTe:SCAN:COUN:SCAN 10 ROUTe:SCAN:INT 1.0 INIT *WAI READ? "defbuffer1" TRAC:DATA? 1, 91, "defbuffer1", READ</pre>	Reset the instrument. Clear defbuffer1 and set it to 100 readings. Set channels 1 to 9 on slot 1 to make DC voltage measurements on the 10 V range at 0.1 PLC. Display 5.5 digits. Create a scan of channels 1 to 9. Set the scan count to 10. Provide a one second delay between each scan. Start the scan. Read and output the data.

Also see

None

:ROUTe:SCAN:COUNT:STEP?

This command returns the number of steps in the present scan.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:ROUTe:SCAN:COUNT:STEP?

Details

This is set by the number of steps in the active scan list.

Example

ROUT:SCAN:COUN:STEP?	Responds with the present step count. Output assuming there are five steps in the scan list: 5

Also see

None

:ROUTe:SCAN[:CREate]

This command deletes the existing scan list and creates a new list of channels to scan.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Not saved Save settings	Empty scan list

Usage

```
:ROUTe:SCAN[:CREate]
:ROUTe:SCAN[:CREate] (@<channelList>)
:ROUTe:SCAN[:CREate] (@<channelList>), "<configurationList>"
:ROUTe:SCAN[:CREate] (@<channelList>), "<configurationList>", <index>
:ROUTe:SCAN[:CREate]?
```

<channelList>	The channels to set, using standard channel naming (on page 11-6); ALLSLOTS and SLOT options not allowed; send (@) to clear the scan list
<configurationList>	A string that defines the configuration list to recall
<index>	The index in the configuration list to recall; default is 1

Details

The items in the channel list are scanned in the order listed.

Sending this command with no parameters clears the existing scan list.

Using a configuration list allows you to set multiple functions for the channels using the settings in the configuration list. The configuration list must exist before you send this command.

NOTE

The front-panel SCAN menu does not show settings set by the configuration list parameter. To check settings, use the command ROUT:SCAN?

The query returns a list of the channels in the scan. If no scan is set up, it returns (@).

Example 1

SENS:FUNC 'VOLT', (@101:110) SENS:VOLT:NPLC 1, (@101:110) SENS:FUNC 'RES', (@111:120) SENS:VOLT:NPLC 1, (@111:120) ROUT:SCAN:CRE (@101,111,120)	This example sets DMM settings on twenty channels and then creates a scan for three of the channels.

Example 2

```

SENS:FUNC 'VOLT', (@101:120)
SENS:VOLT:NPLC 1, (@101:120)

SENS:FUNC 'RES', (@111:120)
SENS:VOLT:NPLC 1, (@111:120)

SENS:CONF:LIST:CRE "dmm_active"
SENS:CONF:LIST:STOR "dmm_active"

ROUT:SCAN:CRE (@101:120)
ROUT:SCAN:ADD (@101:120), "dmm_active"
INIT

```

Set up the DMM on twenty channels.
 Create the configuration list `dmm_active` and stores the DMM settings.
 Create a scan.
 Adds channels to the scan that use the settings stored in `dmm_active`.
 Start the scan.

Example 3

```

SENS:FUNC "VOLT", (@101:109)
SENS:VOLT:NPLC 1, (@101:109)

ROUT:SCAN:CRE (@101:109)

ROUT:SCAN:EXPORT "/usb1/mydata", END,
  ALL

INIT

```

Insert a USB flash drive into the unit.
 This example sets DMM settings on nine channels and then creates a scan. When the scan is run, the data is saved to the USB flash drive.

Example 4

```

*RST
TRAC:CLE
TRACe:POINts 100, "defbuffer1"

ROUT:SCAN (@101:109)
SENS:FUNC "VOLT", (@101:109)
SENS:VOLT:RANG 10, (@101:109)
SENS:VOLT:NPLC 0.1, (@101:109)
DISP:VOLT:DIG 5, (@101:109)

ROUT:SCAN:CRE (@101:109)
ROUTe:SCAN:COUN:SCAN 10
ROUTe:SCAN:INT 1.0

INIT
*WAI

READ? "defbuffer1"
TRAC:DATA? 1, 91, "defbuffer1", READ

```

Reset the instrument.
 Clear `defbuffer1` and set it to 100 readings.
 Set channels 1 to 9 on slot 1 to make DC voltage measurements on the 10 V range at 0.1 PLC.
 Display 5.5 digits.

Create a scan of channels 1 to 9.
 Set the scan count to 10.
 Provide a one second delay between each scan.

Start the scan.

Read and output the data.

Also see

[:ROUTe:SCAN:ADD](#) (on page 12-82)
[:ROUTe:SCAN:ADD:SINGLE](#) (on page 12-84)
[Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:EXPort

This command stores data from a scan to a file on a USB flash drive.

Type	Affected by	Where saved	Default value
Command only	Restore settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:ROUTe:SCAN:EXPort "/usb1/<filename>", <when>
:ROUTe:SCAN:EXPort "/usb1/<filename>", <when>, <what>
```

<filename>	The name of the file to be created on the USB flash drive
<when>	When to write the data to the file: - STEP: At completion of each scan step - SCAN: At completion of each scan - END: At completion of all scans - NEVER: Do not write data to the file
<what>	Which data to include: - All information: ALL - Dates, times, and fractional seconds are saved; the default value: FORMat - Relative timestamps are saved: RELative - Seconds and fractional seconds are saved: RAW - Timestamps are saved: STAMp - Standard set of data: STANdard - Brief set of data (reading and relative timestamp only): BRIEF - Extra data, such as the reading and unit from a DC voltage ratio measurement: EXTRa - Channel information: CHANCOL - Compact information formatted so that it is easy to graph in Microsoft Excel: EASYGRAPH If nothing is defined, the output defaults to the reading, unit, range digits, display digits, status, extra values, and channel

Details

This command sets up the instrument to export scan data. If an option to export data is selected, data is sent to a USB flash drive inserted into the USB port on the front panel of the instrument. Export files are limited to 500 MB. When data exceeds 500 MB, another file is created with `_n` added to the file name, where `n` starts at 1 and is incremented for each additional file.

The file name must specify the full path (including `/usb1/`). If included, the file extension must be set to `.csv`. If no file extension is specified, `.csv` is added.

The exported data is time-stamped.

Exporting data can impact scan performance. The more often exports occur, the more the impact on performance. Therefore, exporting data at completion of each step results in the slowest performance.

The DAQ6510 does not check for existing files when you save. Verify that you are using a unique name to avoid overwriting any existing `.csv` files on the flash drive.

You cannot use `CHANCOL` if `<when>` is set to `STEP`.

You cannot use `EASYGRAPH` if `<when>` is set to `STEP` or `SCAN`.

Example

```
ROUT:SCAN:EXP "/usb1/myData.csv", STEP, STAN
```

Save the scan data to a file named `myData.csv` on the USB flash drive. The data includes the standard set of data for each step of the scan.

Also see

None

:ROUTe:SCAN:INTerval

This command specifies the interval time between scan starts when the scan count is more than one.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (0 s)

Usage

```
:ROUTe:SCAN:INTerval <interval>  
:ROUTe:SCAN:INTerval?
```

<interval>	The scan interval (0 s to 100 ks)

Details

If the scan interval is less than the time the scan takes to run, the next scan starts immediately when the first scan finishes.

Example

ROUT:SCAN:CRE (@101:109) ROUT:SCAN:COUNT:SCAN 10 ROUT:SCAN:INT 1	Create a scan of channels 1 to 9. Set the scan count to 10. Provide a one second delay after each scan.

Also see

[Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:LEARn:LIMits

This command calculates alarm limits based on the present configuration of the system.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
ROUTe:SCAN:LEARn:LIMits <window>
ROUTe:SCAN:LEARn:LIMits <window>, <iterations>
```

<window>	Percentage of deviation from the measurement that is within limits: 0.1 to 1000
<iterations>	The number of times the scan should run to establish the limits: 1 to 10

Details

Auto Learn runs a scan and establishes alarm limits based on the measurements from the scan. Make sure your system is in a stable state before running Auto Learn.

Example

SENS:FUNC "VOLT", (@101:105) ROUT:SCAN:CRE (@101:105) ROUT:SCAN:LEAR:LIM 1, 5	Create a scan on channels 101 to 105. Set up the channels to measure DC voltage. Start the learn limits calculation, with a 1% window and 5 iterations.

Also see

None

:ROUTe:SCAN:MEASure:STIMulus

This command selects the trigger for the measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:ROUTe:SCAN:MEASure:STIMulus <eventID>  
:ROUTe:SCAN:MEASure:STIMulus?
```

<eventID>	The event that triggers the measurement

Details

Use this to start a set of measurement count readings that are triggered by a single event.
The available trigger events are described in the following table.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMManD
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENDer<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPLete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmlimit (returns NOT3)

Example 1

```
*RST
SENS:FUNC 'RES', (@101:109)
ROUT:SCAN:CRE (@101:109)
ROUT:SCAN:COUN:SCAN 10
ROUT:SCAN:MEAS:STIM EXT
TRIG:EXT:IN:EDGE FALL
TRIG:EXT:OUT:LOG NEG
TRIG:EXT:OUT:STIM SCANMEAS
INIT
```

Reset the instrument.
Set channels 101 through 109 to measure 2-wire resistance.
Create a scan using channels 101 through 109.
Set the scan count to 10.
Set the channel measurement stimulus to be triggered by a falling edge pulse on the EXTERNAL TRIGGER IN line.
Set the EXTERNAL TRIGGER OUT line to generate a negative pulse each time a scan channel makes a measurement.
Initiate the scan.

Example 2

<pre>*RST SENS:FUNC 'RES', (@101:109) ROUT:SCAN:CRE (@101:109) ROUT:SCAN:COUN:SCAN 10 ROUT:SCAN:BYPASS ON ROUT:SCAN:CHAN:STIM DIG1 DIG:LINE1:MODE TRIG, IN TRIG:DIG1:IN:EDGE FALL DIG:LINE3:MODE TRIG, OUT TRIG:DIG3:OUT:LOG NEG TRIG:DIG3:OUT:STIM SCANCHAN INIT</pre>	Reset the instrument. Set channels 101 through 109 for 2-wire resistance measurements. Create a scan of channels 101 through 109. Set the scan count to 10. Bypass the trigger for the first channel close. Set the channel close stimulus to respond to a falling edge trigger on digital input line 1. Set a digital output signal to trigger a negative pulse for each time that a defined scan channel is closed. Start the scan.

Also see

[Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:MEASure:INTerval

This command specifies the interval time between measurement requests.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (0 s)

Usage

:ROUTe:SCAN:MEASure:INTerval <time>
:ROUTe:SCAN:MEASure:INTerval?

<time>	The interval time between measurements (0 s to 100 ks)

Details

This command specifies the time between measurements in the scan.

Example

<pre>ROUT:SCAN (@101:109) SENS:FUNC "VOLT", (@101:109) SENS:VOLT:RANG 10, (@101:109) SENS:VOLT:NPLC 0.1, (@101:109) ROUT:SCAN:CRE (@101:109) ROUT:SCAN:COUNt:SCAN 10 ROUT:SCAN:MEAS:INT 1.0 INIT *WAI READ? "defbuffer1" TRAC:DATA? 1, 90, "defbuffer1", READ</pre>	Set channels 1 to 9 on slot 1 to make DC voltage measurements on the 10 V range at 0.1 PLC. Create a scan of channels 1 to 9. Set the scan count to 10. Provide a one second delay between each measurement in the scan. Start the scan. Read and output the data.

Also see

[Scanning and triggering](#) (on page 5-29)
[:ROUTe:SCAN:COUNt:SCAN](#) (on page 12-91)

:ROUTe:SCAN:MODE

This command sets the relay action when the scan starts.

Type	Affected by	Where saved	Default value
Command and query	Restore setup Instrument reset Power cycle	Save settings	ALL

Usage

```
:ROUTe:SCAN:MODE <mode>  
:ROUTe:SCAN:MODE?
```

<mode>	Channel action when the scan starts: ■ ALL: Open all channels on all slots before a scan starts ■ USED: See Details ■ ABR: Automatic backplane relay; see Details

Details

When this attribute is set to open all, all channels on all slots are opened before a scan starts.

When the mode is set to open used, an intelligent open is performed. For channels that are not set to a function:

- All channels used in scanning are opened
- Closed channels not used in scanning remain closed during the scan

If any step is set to a function:

- Any commonside ohms backplane relays are opened on all slots
- Any amp channels are opened on all slots
- All channels and backplane relays involved in scanning are opened
- If a closed channel or backplane relay is not involved in scanning, it remains closed during the scan
- All channels are opened on any bank that contains backplane relays that are involved in scanning

When this attribute is set to automatic backplane relay, it is equivalent to setting open used, except that all required backplane relays are closed before the start of the scan. These backplane relays are not opened or closed during the scan and do not open at the end of the scan.

Example

ROUT:SCAN:MODE USED	Sets the scan mode setting to open only channels that are used in the scan.

Also see

[Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:MONitor:CHANnel

This command defines which channel to monitor for a limit to be reached before starting the scan.

Type	Affected by	Where saved	Default value
Command and query	Restore setup Instrument reset Power cycle	Save settings	101

Usage

```
:ROUTe:SCAN:MONitor:CHANnel (@<channel>)  
:ROUTe:SCAN:MONitor:CHANnel?
```

<channel>	The channel to monitor

Details

The channel to monitor for a limit to be reached before starting the scan.

Example

<pre>SENS:FUNC "VOLT", (@101:105) SENS:VOLT:NPLC 0.01, (@101:105) SENS:FUNC "TEMP", (@201) ROUT:SCAN:MON:CHAN (@201) ROUT:SCAN:MON:LIM:UPP 28 ROUT:SCAN (@101:105) ROUT:SCAN:MON:MODE UPP INIT</pre>	With 7700 switching modules in slot 1 and slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01. Set up channel 1 in slot 2 to measure temperature. Set the monitoring channel to be 201. Set the high limit to 28 °C. Create a scan that includes data for channels 101 to 105. Set the scan monitor function to allow scanning to start when the measured value exceeds the defined high limit value (28 °C). Initiate the scan, which starts when the temperature of channel 201 is above 28 °C.

Also see

- [:ROUTe:SCAN:MONitor:LIMit:LOWer\[:DATA\]](#) (on page 12-105)
- [:ROUTe:SCAN:MONitor:LIMit:UPPer\[:DATA\]](#) (on page 12-107)
- [:ROUTe:SCAN:MONitor:MODE](#) (on page 12-106)

:ROUTe:SCAN:MONitor:LIMit:LOWer[:DATA]

This command defines the low limit to be used by the scan monitor.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:ROUTe:SCAN:MONitor:LIMit:LOWer[:DATA] <n>  
:ROUTe:SCAN:MONitor:LIMit:LOWer[:DATA]?
```

<n>	The value of the lower limit applied to the monitor channel

Details

This command sets the low limit for the monitor.

Example

```
SENS:FUNC "VOLT", (@101:105)  
SENS:VOLT:NPLC 0.01, (@101:105)  
SENS:FUNC "VOLT:AC", (@201:202)  
ROUT:SCAN:MON:LIM:LOW -3  
ROUT:SCAN:MON:LIM:UPP 1  
ROUT:SCAN:CREATe (@101:105, 201, 202)  
ROUT:SCAN:MON:MODE WIND  
ROUT:SCAN:MON:CHAN (@101)  
INIT
```

With a 7701 in slot 1 and a 7708 in slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01. Set up channels 1 and 2 on slot 2 to measure AC voltage.

Set the low limit for the monitor scan to -3 and the high limit to 1.

Create a scan that includes channels 101 to 105 and 201 to 202.

Set the scan to start when the limits are outside a set of values.

Monitor channel 101.

Initiate the scan. The scan starts when voltage of channel 101 is below -3 or above 1.

Also see

- [:ROUTe:SCAN:MONitor:CHANnel](#) (on page 12-104)
- [:ROUTe:SCAN:MONitor:LIMit:UPPer\[:DATA\]](#) (on page 12-107)
- [:ROUTe:SCAN:MONitor:MODE](#) (on page 12-106)

:ROUTe:SCAN:MONitor:MODE

This command determines if a scan starts immediately when triggered or after measurements reach a set value.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	OFF

Usage

```
:ROUTe:SCAN:MONitor:MODE <n>
:ROUTe:SCAN:MONitor:MODE?
```

<n>

Which value to use to start the scan:

- **OFF**: Start the scan without waiting for a specific value
- **UPPer**: Start the scan when the measurement exceeds the value set by :ROUTe:SCAN:MONitor:LIMit:UPPer[:DATA]
- **LOWer**: Start the scan when the measurement is below the value set by :ROUTe:SCAN:MONitor:LIMit:LOWer[:DATA]
- **OUTSide**: Start the scan when the measurement is less than the lower limit or more than the upper limit.
- **WINDow**: Start the scan when the measurement is between the lower limit and the upper limit

Details

This command determines if measurements are monitored to start a scan. If measurements are monitored, it also determines if the measurement triggers the start of the scan when it reaches a high value, low value, inside the high and low values, or outside the high and low values. The scan start values are stored in defbuffer2.

Example

```
SENS:FUNC "VOLT", (@101:105)
SENS:VOLT:NPLC 0.01, (@101:105)
SENS:FUNC "VOLT:AC", (@201:202)
ROUT:SCAN:MON:LIM:LOW -3
ROUT:SCAN:MON:LIM:UPP 1
ROUT:SCAN:CREATe (@101:105, 201, 202)
ROUT:SCAN:MON:MODE WIND
ROUT:SCAN:MON:CHAN (@101)
INIT
```

With a 7701 in slot 1 and a 7708 in slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01. Set up channels 1 and 2 on slot 2 to measure AC voltage.

Set the low limit for the monitor scan to -3 and the high limit to 1.

Create a scan that includes channels 101 to 105 and 201 to 202.

Set the scan to start when the limits are outside a set of values.

Monitor channel 101.

Initiate the scan. The scan starts when voltage of channel 101 is below -3 or above 1.

Also see

[:ROUTe:SCAN:MONitor:CHANnel](#) (on page 12-104)

[:ROUTe:SCAN:MONitor:LIMit:LOWer\[:DATA\]](#) (on page 12-105)

[:ROUTe:SCAN:MONitor:LIMit:UPPer\[:DATA\]](#) (on page 12-107)

:ROUTe:SCAN:MONitor:LIMit:UPPer[:DATA]

This command specifies the high limit to be used by the scan monitor

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:ROUTe:SCAN:MONitor:LIMit:UPPer[:DATA] <n>
:ROUTe:SCAN:MONitor:LIMit:UPPer[:DATA] ?
```

<n>	The value of the upper limit applied to the monitor channel

Details

This command sets the high limit for the monitor.

Example

```
SENS:FUNC "VOLT", (@101:105)
SENS:VOLT:NPLC 0.01, (@101:105)
SENS:FUNC "VOLT:AC", (@201:202)
ROUT:SCAN:MON:LIM:LOW -3
ROUT:SCAN:MON:LIM:UPP 1
ROUT:SCAN:CREATE (@101:105, 201, 202)
ROUT:SCAN:MON:MODE WIND
ROUT:SCAN:MON:CHAN (@101)
INIT
```

With a 7701 in slot 1 and a 7708 in slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01. Set up channels 1 and 2 on slot 2 to measure AC voltage.

Set the low limit for the monitor scan to -3 and the high limit to 1.

Create a scan that includes channels 101 to 105 and 201 to 202.

Set the scan to start when the limits are outside a set of values.

Monitor channel 101.

Initiate the scan. The scan starts when voltage of channel 101 is below -3 or above 1.

Also see

[:ROUTe:SCAN:MONitor:CHANnel](#) (on page 12-104)

[:ROUTe:SCAN:MONitor:LIMit:LOWer\[:DATA\]](#) (on page 12-105)

[:ROUTe:SCAN:MONitor:MODE](#) (on page 12-106)

:ROUTe:SCAN:REStart

This command causes a scan to automatically restart if it was interrupted by a power failure.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	OFF

Usage

:ROUTe:SCAN:REStart <n>
:ROUTe:SCAN:REStart?

<n>	OFF: Disable scan restart ON: Enable scan restart

Details

If the restart option is enabled, the scan settings are saved in memory immediately after the scan is triggered and before the scan operation begins. All scan settings, including watched channels, need to be set before the scan starts. Any changes that are made after the scan starts are not recalled if power is lost and the scan needs to restart.

If the restart option is enabled and power is lost, the scan restarts when power is restored. The scan setup that was in place when the scan started becomes the power-up setup. It takes precedence over any other power-up setup. If the scan completes successfully, the scan setup is removed as the power-up setup.

If the DAQ6510 detects that a card was changed during the power-up sequence, restart is set to off, the interrupted scan is not resumed, and an event is generated. The instrument starts up normally.

When a scan is automatically restarted, it is logged in the event log.

NOTE

If a restart occurs, the original reading buffer data is retained (per the settings of the reading buffer) and a new reading buffer is created.

Example

ROUT:SCAN:REST ON	Set scan to restart when power to the instrument is restored.

Also see

None

:ROUTe:SCAN:START:STIMulus

This command determines which event starts the scan.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:ROUTe:SCAN:START:STIMulus <eventID>  
:ROUTe:SCAN:START:STIMulus?
```

<eventID>	Trigger stimulus used to start the scan; see Details

Details

The events that you can use to start the scan are described in the following table.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENder<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPlete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmlimit (returns NOT3)

Example 1

ROUT:SCAN:STAR:STIM SCANCHAN

Start the scan when the channels have closed.

Example 2

ROUT:SCAN:STAR:STIM NONE

Start the scan immediately.

Example 3

ROUT:SCAN:STAR:STIM DIG3

The scan begins when the instrument receives a signal from digital I/O line 3.

Also see[Scanning and triggering](#) (on page 5-29)

:ROUTe:SCAN:STATe?

This command provides the present state of a running background scan.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:ROUTe:SCAN:STATe?

Details

Returns the state of the present scan, the scan count, and the step count.

- The scan count is the number of the present iteration through the scan portion of the trigger model. This number does not increment until the scan begins. Therefore, if the instrument is waiting for an input to trigger a scan start, the scan count represents the previous number of scan iterations. If no scan has begun, the scan count is zero.
- The step count is 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 instrument is waiting for an input to trigger a channel action, the step count represents the previous step. If no step has yet completed, the step count is 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.
- The information from the scan state command may be delayed up to 100 ms from the actual state of the scan because of system resources used by the scan.

Scans are based on trigger models, so the states are reported as trigger model states. The trigger model states are:

- **Idle:** The trigger model is stopped
- **Running:** The trigger model is running
- **Waiting:** The trigger model has been in the same wait block for more than 100 ms
- **Empty:** The trigger model is selected, but no blocks are defined
- **Paused:** The trigger model is paused
- **Building:** Blocks have been added
- **Failed:** The trigger model is stopped because of an error
- **Aborting:** The trigger model is stopping
- **Aborted:** The trigger model is stopped
- **Success:** The scan completed successfully.

Example

```
ROUT:SCAN:STAT?
```

If scan is running, output is similar to:
RUNNING;1;2

Also see

None

:ROUTe:TERMinals?

This command describes which set of input and output terminals the instrument is using.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:ROUTe:TERMinals?

Details

You must use the front-panel TERMINALS switch to change which set of terminals the instrument reads.

This query returns the set of input and output terminals that the instrument is using. If the instrument is using the front-panel terminals, the return is:

FRON

If the instrument is using the rear-panel terminals, the return is:

REAR

Example

:ROUT:TERM?	Query to verify which terminals are used. Output if the rear terminals are used: REAR

Also see

None

SCRipt subsystem

The SCRipt subsystem controls macro or instrument setup scripts. For additional information on macro scripts, refer to [Saving front-panel settings into a macro script](#) (on page 4-97).

:ScripT:RUN

This command runs a script.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

ScripT:RUN "<scriptName>"

<scriptName>	The name of the script

Details

The script must be available in the instrument to be used by this command.

Example

SCR:RUN "bufferCreate"	Runs a script named bufferCreate.

Also see

- [Saving front-panel settings into a macro script](#) (on page 4-97)
- [Scripts menu](#) (on page 3-63)

SENSe1 subsystem

The SENSe1 subsystem commands configure and control the measurement functions of the instrument.

Many of these commands are set for a specific function. For example, you can program a range setting for each function. The settings are saved with that function.

[[:SENSe[1]]]:<function>:APERture

This command determines the aperture setting for the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	See Details

Usage

```
[[:SENSe[1]]]:<function>:APERture <n>
[:SENSe[1]]:<function>:APERture <DEF|MIN|MAX>
[:SENSe[1]]:<function>:APERture <n>, (@<channelList>)
[:SENSe[1]]:<function>:APERture <DEF|MIN|MAX>, (@<channelList>)
[:SENSe[1]]:<function>:APERture?
[:SENSe[1]]:<function>:APERture? <DEF|MIN|MAX>
[:SENSe[1]]:<function>:APERture? (@<channelList>)
[:SENSe[1]]:<function>:APERture? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The time of the aperture; see Details
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Function	Default value	Range
Voltage (AC and DC)	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Current (AC and DC)	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Resistance (2-wire and 4-wire)	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Diode	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Temperature	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Frequency and Period	200 ms	2 ms to 273 ms
Voltage ratio	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s

Digitize (voltage and current)	AUTO	1 μ s to 1 ms set in 1 μ s increments

The functionality of aperture depends on whether you are using a measure function or a digitize function.

Aperture for a measure function

If you are using a measure function, the aperture sets the amount of time the ADC takes when making a measurement, which is the integration period for the selected measurement function. The integration period is specified in seconds. In general, a short integration period provides a fast reading rate, while a long integration period provides better accuracy. The selected integration period is a compromise between speed and accuracy.

During the integration period, if an external trigger with a count of 1 is sent, the trigger is ignored. If the count is set to more than 1, the first reading is initialized by this trigger. Subsequent readings occur as rapidly as the instrument can make them. If a trigger occurs during the group measurement, the trigger is latched and another group of measurements with the same count will be triggered after the current group completes.

You can also set the integration rate by setting the number of power-line cycles (NPLCs). Changing the NPLC value changes the aperture time and changing the aperture time changes the NPLC value.

To calculate the aperture based on the NPLC value, use the following formula.

$$\text{Aperture} = \frac{\text{NPLC}}{f}$$

where:

- Aperture is the integration rate in seconds for each integration
- NPLC is the number of power-line cycles for each integration
- f is the power-line frequency

If you set the NPLCs, the aperture setting changes to reflect that value. If you set the aperture, the NPLC setting is changed.

For the AC voltage and AC current functions, the aperture value is fixed and cannot be changed.

If line synchronization is enabled, the integration period does not start until the beginning of the next power-line cycle. For example, if a reading is triggered at the positive peak of a power-line cycle, the integration period does not start until that power-line cycle is completed. The integration period starts when the positive-going sine wave crosses 0 volts.

To see the line frequency that is automatically detected by the instrument, use the `:SYSTem:LFRequency?` command.

Aperture for a digitize function

If you are using a digitize function, the aperture is the actual acquisition time of the instrument on the signal. The aperture can be set to automatic or to a specific value in 1 μ s intervals. If the value is not specified in microseconds, the value is rounded down to the nearest microsecond resolution. When automatic is selected, the aperture setting is set to the maximum value possible for the selected sample rate.

The aperture must be less than the reciprocal of the sample rate. The minimum aperture is 1 μ s at the maximum sampling rate of 1,000,000 samples per second.

Set the sample rate before changing the aperture.

The maximum aperture available is 1 divided by the sample rate. The aperture cannot be set to more than this value. You select automatic by sending `AUTO`.

Example

```
DIG:FUNC "CURR"  
DIG:CURR:SRATE 1000000  
DIG:CURR:APER AUTO  
DIG:COUN 10  
MEAS:DIG?
```

Set the digitize function to measure current. Set the sample rate to 1,000,000, with a count of 10, and automatic aperture. Make a digitize measurement.

Also see

[\[:SENSe\[1\]\]:<function>:NPLCycles](#) (on page 12-143)

[\[:SENSe\[1\]\]:<function>:SRATe](#) (on page 12-165)

[:SYSTem:LFRequency?](#) (on page 12-224)

[[:SENSE[1]]]:<function>:ATrigger:EDGE:LEVel

This command defines the signal level that generates the analog trigger event for the edge trigger mode.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0

Usage

```
[[:SENSE[1]]]:<function>:ATrigger:EDGE:LEVel <setting>
[[:SENSE[1]]]:<function>:ATrigger:EDGE:LEVel <setting>, (@<channelList>)
[[:SENSE[1]]]:<function>:ATrigger:EDGE:LEVel?
[[:SENSE[1]]]:<function>:ATrigger:EDGE:LEVel? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<setting>	The signal level that generates the trigger event
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURREnt
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command is only available when the analog trigger mode is set to edge.

The edge level can be set to any value in the active measurement range.

To use the analog trigger with the measure functions, a range must be set (you cannot use autorange) and autozero must be disabled.

Example

<pre>FUNC "CURR" CURR:RANGE 3 CURR:AZER OFF CURR:ATR:MODE EDGE CURR:ATR:EDGE:LEV 2.5</pre>	<pre>Set measure function to DC current. Set range to 3 A. Disable autozero. Set the analog trigger mode to edge. Set the analog trigger level to 2.5 A.</pre>

Also see

[Analog triggering overview](#) (on page 8-20)
[\[:SENSE\[1\]\]:<function>:ATrigger:MODE](#) (on page 12-121)
[\[:SENSE\[1\]\]:<function>:AZERo\[:STATe\]](#) (on page 12-133)
[\[:SENSE\[1\]\]:<function>:RANGe\[:UPPer\]](#) (on page 12-149)

[[:SENSe[1]]]:<function>:ATRigger:EDGE:SLOPe

This command defines the slope of the analog trigger edge.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	RISing

Usage

```
[[:SENSe[1]]]:<function>:ATRigger:EDGE:SLOPe <setting>
[:SENSe[1]]:<function>:ATRigger:EDGE:SLOPe?
[:SENSe[1]]:<function>:ATRigger:EDGE:SLOPe <setting>, (@<channelList>)
[:SENSe[1]]:<function>:ATRigger:EDGE:SLOPe? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<setting>	The direction: ■ Rising: RISing ■ Falling: FALLing
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This is only available when the analog trigger mode is set to edge.

Rising causes an analog trigger event when the analog signal trends from below the analog signal level to above the level.

Falling causes an analog trigger event when the signal trends from above to below the level.

Example

<pre>FUNC "CURR" CURR:RANGE 3 CURR:AZER OFF CURR:ATR:MODE EDGE CURR:ATR:EDGE:LEV 2.5 CURR:ATR:EDGE:SLOP RIS</pre>	<pre>Set measure function to DC current. Set range to 3 A. Disable autozero. Set the analog trigger mode to edge. Set the analog trigger level to 2.5 A. Set the analog trigger slope to rising.</pre>

Also see

[\[:SENSe\[1\]\]:<function>:ATRigger:EDGE:LEVel](#) (on page 12-119)

[\[:SENSe\[1\]\]:<function>:ATRigger:MODE](#) (on page 12-121)

[Analog triggering overview](#) (on page 8-20)

[:SENSe[1]] :<function> :ATRigger :MODE

This command configures the type of signal behavior that can generate an analog trigger event.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF

Usage

```
[ :SENSe[1]] :<function> :ATRigger :MODE <setting>
[ :SENSe[1]] :<function> :ATRigger :MODE?
[ :SENSe[1]] :<function> :ATRigger :MODE <setting>, (@<channelList>)
[ :SENSe[1]] :<function> :ATRigger :EDGE :MODE? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<setting>	The setting: - Edge (signal crosses one level): EDGE - Window (signal enters or exits a window defined by two levels): WINDow - No analog triggering: OFF
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

When edge is selected, the analog trigger occurs when the signal crosses a certain level. You also specify if the analog trigger occurs on the rising or falling edge of the signal.

When window is selected, the analog trigger occurs when the signal enters or exits the window defined by the low and high signal levels.

Example

<pre>FUNC "CURR" CURR:RANGE 3 CURR:AZER OFF CURR:ATR:MODE EDGE CURR:ATR:EDGE:LEV 2.5 CURR:ATR:EDGE:SLOP RIS</pre>	<pre>Set measure function to DC current. Set range to 3 A. Disable autozero. Set the analog trigger mode to edge. Set the analog trigger level to 2.5 A. Set the analog trigger slope to rising.</pre>

Also see

[Analog triggering overview](#) (on page 8-20)

[[:SENSE[1]]]:<function>:ATRigger:WINDow:DIRection

This command defines if the analog trigger occurs when the signal enters or leaves the defined high and low analog signal level boundaries.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ENTer

Usage

```
[[:SENSE[1]]]:<function>:ATRigger:WINDow:DIRection <setting>
[:SENSE[1]]:<function>:ATRigger:WINDow:DIRection?
[:SENSE[1]]:<function>:ATRigger:WINDow:DIRection <setting>, (@<channelList>)
[:SENSE[1]]:<function>:ATRigger:WINDow:DIRection? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<setting>	The direction: ■ Enter: ENTer ■ Leave: LEAVe
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This is only available when the analog trigger mode is set to window.

Example

<pre>FUNC "CURR" CURR:RANGE 3 CURR:AZER OFF CURR:ATR:MODE WINDOW CURR:ATR:WIND:LEV:HIGH 2.5 CURR:ATR:WIND:LEV:LOW 1 CURR:ATR:WIND:DIR LEAV</pre>	Set measure function to DC current. Set range to 3 A. Disable autozero. Set the analog trigger mode to window. Set the analog trigger level for the low point of the window to 1.0 A and the high point for 2.5 A. Set the trigger to occur when the signal leaves the window (signal below 1.0 A or above 2.5 A).

Also see

[Analog triggering overview](#) (on page 8-20)

[\[:SENSE\[1\]\]:<function>:ATRigger:MODE](#) (on page 12-121)

[\[:SENSE\[1\]\]:<function>:ATRigger:WINDow:LEVel:HIGH](#) (on page 12-123)

[\[:SENSE\[1\]\]:<function>:ATRigger:WINDow:LEVel:LOW](#) (on page 12-124)

[[:SENSe[1]]]:<function>:ATRigger:WINDow:LEVel:HIGH

This command defines the upper boundary of the analog trigger window.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	DC current and digitize current: 5e-6 DC voltage and digitize voltage: 0.05

Usage

```
[[:SENSe[1]]]:<function>:ATRigger:WINDow:LEVel:HIGH <value>
[:SENSe[1]]:<function>:ATRigger:WINDow:LEVel:HIGH?
[:SENSe[1]]:<function>:ATRigger:WINDow:LEVel:HIGH <setting>, (@<channelList>)
[:SENSe[1]]:<function>:ATRigger:WINDow:LEVel:HIGH? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<value>	The upper boundary of the window
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Only available when the analog trigger mode is set to window.

The high level must be greater than the low level.

To use the analog trigger with the measure functions, a range must be set (you cannot use autorange) and autozero must be disabled.

Example

<pre>FUNC "CURR" CURR:RANGE 3 CURR:AZER OFF CURR:ATR:MODE WINDOW CURR:ATR:WIND:LEV:HIGH 2.5 CURR:ATR:WIND:LEV:LOW 1 CURR:ATR:WIND:DIR LEAV</pre>	Set measure function to DC current. Set range to 3 A. Disable autozero. Set the analog trigger mode to window. Set the analog trigger level for the low point of the window to 1.0 A and the high point for 2.5 A. Set the trigger to occur when the signal leaves the window (signal below 1.0 A or above 2.5 A).

Also see

[Analog triggering overview](#) (on page 8-20)

[\[:SENSe\[1\]\]:<function>:ATRigger:MODE](#) (on page 12-121)

[\[:SENSe\[1\]\]:<function>:ATRigger:WINDow:DIRection](#) (on page 12-122)

[\[:SENSe\[1\]\]:<function>:ATRigger:WINDow:LEVel:LOW](#) (on page 12-124)

[:SENSe[1]] :<function> :ATRigger :WINDow :LEVel :LOW

This command defines the lower boundary of the analog trigger window.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0

Usage

```
[ :SENSe[1]] :<function> :ATRigger :WINDow :LEVel :LOW <value>
[ :SENSe[1]] :<function> :ATRigger :WINDow :LEVel :LOW?
[ :SENSe[1]] :<function> :ATRigger :WINDow :LEVel :LOW <setting>, (@<channelList>)
[ :SENSe[1]] :<function> :ATRigger :WINDow :LEVel :LOW? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<value>	The lower boundary of the window
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTInuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Only available when the analog trigger mode is set to window.

The low level must be less than the high level.

To use the analog trigger with the measure functions, a range must be set (you cannot use autorange) and autozero must be disabled.

Example

<pre>FUNC "CURR" CURR:RANGE 3 CURR:AZER OFF CURR:ATR:MODE WINDOW CURR:ATR:WIND:LEV:HIGH 2.5 CURR:ATR:WIND:LEV:LOW 1 CURR:ATR:WIND:DIR LEAV</pre>	<pre>Set measure function to DC current. Set range to 3 A. Disable autozero. Set the analog trigger mode to window. Set the analog trigger level for the low point of the window to 1.0 A and the high point for 2.5 A. Set the trigger to occur when the signal leaves the window (signal below 1.0 A or above 2.5 A).</pre>

Also see

[Analog triggering overview](#) (on page 8-20)

[\[:SENSe\[1\]\] :<function> :ATRigger :MODE](#) (on page 12-121)

[\[:SENSe\[1\]\] :<function> :ATRigger :WINDow :DIRection](#) (on page 12-122)

[\[:SENSe\[1\]\] :<function> :ATRigger :WINDow :LEVel :HIGH](#) (on page 12-123)

[[:SENSe[1]]]:<function>:AVERage:COUNT

This command sets the number of measurements that are averaged when filtering is enabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	10

Usage

```
[[:SENSe[1]]]:<function>:AVERage:COUNT <n>
[[:SENSe[1]]]:<function>:AVERage:COUNT <DEF|MIN|MAX>
[[:SENSe[1]]]:<function>:AVERage:COUNT <n>, (@<channelList>)
[[:SENSe[1]]]:<function>:AVERage:COUNT <DEF|MIN|MAX>, (@<channelList>)
[[:SENSe[1]]]:<function>:AVERage:COUNT?
[[:SENSe[1]]]:<function>:AVERage:COUNT? <DEF|MIN|MAX>
[[:SENSe[1]]]:<function>:AVERage:COUNT? (@<channelList>)
[[:SENSe[1]]]:<function>:AVERage:COUNT? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions

<n>	The number of readings required for each filtered measurement (1 to 100)

<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value

<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The filter count is the number of readings that are acquired and stored in the filter stack for the averaging calculation. When the filter count is larger, more filtering is done, and the data is less noisy.

Example 1

CURR:AVER:COUNT 10 CURR:AVER:TCON MOV CURR:AVER ON	For current measurements, set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.

Example 2

RES:AVER:COUNT 10 RES:AVER:TCON MOV RES:AVER ON	For resistance measurements, set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.

Example 3

```
VOLT:AVER:COUNT 10  
VOLT:AVER:TCON MOV  
VOLT:AVER ON
```

For voltage measurements, set the averaging filter type to moving average, with a filter count of 10.
Enable the averaging filter.

Also see

[Filtering measurement data](#) (on page 4-65)

[\[:SENSe\[1\]\]:<function>:AVERage\[:STATe\]](#) (on page 12-127)

[\[:SENSe\[1\]\]:<function>:AVERage:TCONtrol](#) (on page 12-129)

[:SENSe[1]] :<function> :AVERage[:STATe]

This command enables or disables the averaging filter for measurements of the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
[ :SENSe[1]] :<function> :AVERage[ :STATe] <state>
[ :SENSe[1]] :<function> :AVERage[ :STATe] <state>, (@<channelList>)
[ :SENSe[1]] :<function> :AVERage[ :STATe]?
[ :SENSe[1]] :<function> :AVERage[ :STATe]? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions

<state>	The filter status; set to one of the following values: - Disable the averaging filter: OFF or 0 - Enable the averaging filter: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command enables or disables the averaging filter. When this is enabled, the reading returned by the instrument is an averaged value, taken from multiple measurements. The settings of the filter count and filter type for the selected measure function determines how the reading is averaged.

Example 1

CURR:AVER:COUNT 10 CURR:AVER:TCON MOV CURR:AVER ON	Set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.

Example 2

RES:AVER:COUNT 10 RES:AVER:TCON MOV RES:AVER ON	Set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.

Example 3

VOLT:AVER:COUNT 10 VOLT:AVER:TCON MOV VOLT:AVER ON	Set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.

Also see

[Filtering measurement data](#) (on page 4-65)

[\[:SENSe\[1\]\]:<function>:AVERage:COUNt](#) (on page 12-125)

[\[:SENSe\[1\]\]:<function>:AVERage:TCONtrol](#) (on page 12-129)

[\[:SENSe\[1\]\]:<function>:AVERage:WINDow](#) (on page 12-131)

[[:SENSe[1]]]:<function>:AVERage:TCONtrol

This command sets the type of averaging filter that is used for the selected measure function when the measurement filter is enabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	REP

Usage

```
[[:SENSe[1]]]:<function>:AVERage:TCONtrol <type>
[:SENSe[1]]:<function>:AVERage:TCONtrol <type>, (@<channelList>)
[:SENSe[1]]:<function>:AVERage:TCONtrol?
[:SENSe[1]]:<function>:AVERage:TCONtrol? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions

<type>	The filter type to use when filtering is enabled; set to one of the following values: - Repeating filter: REPeat - Moving filter: MOVing (not available for channels) - Hybrid filter: HYBRid (only available if the buffer style is set to FULL)
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODe	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command selects the type of averaging filter: Repeating average, hybrid average, or moving average.

When the repeating average filter is selected, a set of measurements are made. These measurements are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed, and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack must be completely filled before an averaged sample can be produced, but it provides more stable results.

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

NOTE

When the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements. The size of the stack is determined by the filter count setting.

The repeating average filter produces slower results but produces more stable results than the moving average filter. For either method, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

The hybrid average filter is only available when the buffer style is set to Full. It is similar to the moving average filter, except that it adds the number of measurements defined by the count to the stack before making the first averaged measurement. This ensures that the filter buffer is filled before returning the first measurement.

The repeating average filter is the only filter option available for use with channels.

Example 1

```
CURR:AVER:COUNT 10
CURR:AVER:TCON MOV
CURR:AVER ON
```

Set the averaging filter type to moving average, with a filter count of 10.
Enable the averaging filter.

Example 2

```
RES:AVER:COUNT 10
RES:AVER:TCON REP
RES:AVER ON
```

Set the averaging filter type to repeating average, with a filter count of 10.
Enable the averaging filter.

Example 3

```
VOLT:AVER:COUN 10, (@101:103)
VOLT:AVER:TCON REP, (@101:103)
VOLT:AVER ON, (@101:103)
```

For scanned voltage measurements on channels 101 through 103, set the averaging filter type to repeating average with a filter count of 10.
Enable the averaging filter.

Also see

[Filtering measurement data](#) (on page 4-65)

[\[:SENSe\[1\]\]:<function>:AVERage:COUNt](#) (on page 12-125)

[\[:SENSe\[1\]\]:<function>:AVERage\[:STATe\]](#) (on page 12-127)

[[:SENSe[1]]]:<function>:AVERage:WINDow

This command sets the window for the averaging filter that is used for measurements for the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (no filter)

Usage

```
[[:SENSe[1]]]:<function>:AVERage:WINDow <n>
[:SENSe[1]]:<function>:AVERage:WINDow <DEF|MIN|MAX>
[:SENSe[1]]:<function>:AVERage:WINDow <n>, (@<channelList>)
[:SENSe[1]]:<function>:AVERage:WINDow <DEF|MIN|MAX>, (@<channelList>)
[:SENSe[1]]:<function>:AVERage:WINDow?
[:SENSe[1]]:<function>:AVERage:WINDow? <DEF|MIN|MAX>
[:SENSe[1]]:<function>:AVERage:WINDow? (@<channelList>)
[:SENSe[1]]:<function>:AVERage:WINDow? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The filter window setting; the range is between 0 and 10 to indicate percent of range
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command selects the window size for the averaging filter.

The noise window allows a faster response time to large signal step changes. A reading that falls outside the plus or minus noise window fills the filter stack immediately.

If the noise does not exceed the selected percentage of range, the reading is based on an average of reading conversions, which is the normal averaging filter. If the noise does exceed the selected percentage, the reading is a single reading conversion, and new averaging starts from this point.

Example

```
CURR:AVER:COUNT 10
CURR:AVER:TCON MOV
CURR:AVER:WIND 5
CURR:AVER ON
```

Set the averaging filter type to moving average, with a filter count of 10 with a window of 5%.
Enable the averaging filter.

Also see

[Filtering measurement data](#) (on page 4-65)

[\[:SENSe\[1\]\]:<function>:AVERage:COUNt](#) (on page 12-125)

[\[:SENSe\[1\]\]:<function>:AVERage\[:STATe\]](#) (on page 12-127)

[\[:SENSe\[1\]\]:<function>:AVERage:TCONtrol](#) (on page 12-129)

[:SENSe[1]] :<function> :AZERo[:STATe]

This command enables or disables automatic updates to the internal reference measurements (autozero) of the instrument.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ON (1)

Usage

```
[ :SENSe[1]] :<function> :AZERo[ :STATe] <state>
[ :SENSe[1]] :<function> :AZERo[ :STATe] <state>, (@<channelList>)
[ :SENSe[1]] :<function> :AZERo[ :STATe]?
[ :SENSe[1]] :<function> :AZERo[ :STATe]? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions

<state>	The status of autozero: - Disable autozero: OFF or 0 - Enable autozero: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODe	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The DAQ6510 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made.

The time to make the reference measurements is in addition to the normal measurement time. If timing is critical, you can disable autozero to avoid this time penalty.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example

VOLT:AZER OFF	Sets autozero off for voltage measurements.

Also see

[Automatic reference measurements](#) (on page 4-53)

[\[:SENSe\[1\]\]:AZERo:ONCE](#) (on page 12-177)

[[:SENSe[1]]]:<function>:BIAS:LEVel

This command selects the amount of current the instrument sources when it makes measurements.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1 mA

Usage

```
[[:SENSe[1]]]:<function>:BIAS:LEVel <n>
[:SENSe[1]]:<function>:BIAS:LEVel <DEF|MIN|MAX>
[:SENSe[1]]:<function>:BIAS:LEVel <n>, (@<channelList>)
[:SENSe[1]]:<function>:BIAS:LEVel <DEF|MIN|MAX>, (@<channelList>)
[:SENSe[1]]:<function>:BIAS:LEVel?
[:SENSe[1]]:<function>:BIAS:LEVel? <DEF|MIN|MAX>
[:SENSe[1]]:<function>:BIAS:LEVel? (@<channelList>)
[:SENSe[1]]:<function>:BIAS:LEVel? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	Enter the value: 10 μA: 1e-5 100 μA: 0.0001 1 mA: 0.001 10 mA: 0.01
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Selects the amount of current that is sourced by the instrument to make measurements.

Example

```
DIOD:BIAS:LEVel 0.0001
```

For the diode functions, sets a bias level of 100 μ A.

Also see

None

[:SENSe[1]] :<function> :DB:REFeRence

This command defines the decibel (dB) reference setting for the DMM in volts.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
[ :SENSe[1]] :<function> :DB:REFeRence <n>
[ :SENSe[1]] :<function> :DB:REFeRence <DEF|MIN|MAX>
[ :SENSe[1]] :<function> :DB:REFeRence <n>, (@<channelList>)
[ :SENSe[1]] :<function> :DB:REFeRence <DEF|MIN|MAX>, (@<channelList>)
[ :SENSe[1]] :<function> :DB:REFeRence?
[ :SENSe[1]] :<function> :DB:REFeRence? <DEF|MIN|MAX>
[ :SENSe[1]] :<function> :DB:REFeRence? (@<channelList>)
[ :SENSe[1]] :<function> :DB:REFeRence? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The decibel reference range: - DC voltage and digitize voltage: 1e-7 V to 1000 V - AC voltage: 1e-7 V to 750 V
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTInuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This value only applies when the unit setting for the function is set to decibels.

Example 1

FUNC "VOLT" VOLT:UNIT DB VOLT:DB:REF 5	Sets the units to decibel and sets the dB reference to 5 for DC volts.

Example 2

FUNC "VOLT:AC", (@101) VOLT:AC:UNIT DB, (@101) VOLT:AC:DB:REF 5, (@101) VOLT:AC:DB:REF? (@101)	On channel 1 in slot one, sets the units to decibel and sets the dB reference to 5 for AC volts. The return from the query is 5.

Also see

[\[:SENSe\[1\]\]:<function>:UNIT](#) (on page 12-176)

[[:SENSe[1]]]:<function>:DBM:REference

This command defines the decibel-milliwatts (dBm) reference.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
[[:SENSe[1]]]:<function>:DBM:REference <n>
[:SENSe[1]]:<function>:DBM:REference <DEF|MIN|MAX>
[:SENSe[1]]:<function>:DBM:REference <n>, (@<channelList>)
[:SENSe[1]]:<function>:DBM:REference <DEF|MIN|MAX>, (@<channelList>)
[:SENSe[1]]:<function>:DBM:REference?
[:SENSe[1]]:<function>:DBM:REference? <DEF|MIN|MAX>
[:SENSe[1]]:<function>:DBM:REference? (@<channelList>)
[:SENSe[1]]:<function>:DBM:REference? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The decibel-milliwatts range (1 to 9999)
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This value only applied when the unit setting for the function is set to dBm.

Example 1

FUNC "VOLT" VOLT:UNIT DBM VOLT:DBM:REF 80	Sets the units to dBm and sets the dBm reference to 80 Ω for DC volts.

Example 2

FUNC "VOLT:AC", (@101) VOLT:AC:UNIT DBM, (@101) VOLT:AC:DBM:REF 5, (@101) VOLT:AC:DBM:REF? (@101)	On channel 1 in slot one, this example sets the units to dBm and sets the dBm reference to 5 Ω for AC volts. The return from the query is 5.

Also see

[\[:SENSe\[1\]\]:<function>:UNIT](#) (on page 12-176)

[:SENSe[1]] :<function> :DELay :AUTO

This command enables or disables the automatic delay that occurs before each measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ON

Usage

```
[ :SENSe[1]] :<function> :DELay :AUTO <state>
[ :SENSe[1]] :<function> :DELay :AUTO <state>, (@<channelList>)
[ :SENSe[1]] :<function> :DELay :AUTO?
[ :SENSe[1]] :<function> :DELay :AUTO? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<state>	Disable the auto delay: OFF Enable the auto delay: ON
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

When this is enabled, a delay is added after a range or function change to allow the instrument to settle.

Example

CURR:DEL:AUTO OFF	Turn off auto delay when DC current is measured.

Also see

[\[:SENSe\[1\]\] :<function> :DELay :USER<n>](#) (on page 12-139)

[:SENSe[1]] :<function> :DELay :USER<n>

This command sets a user-defined delay that you can use in the trigger model.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list Function change	Save settings Measure configuration list	0 (0 s)

Usage

```
[ :SENSe[1]] :<function> :DELay :USER<n> <delayTime>
[ :SENSe[1]] :<function> :DELay :USER<n> <DEF|MIN|MAX>
[ :SENSe[1]] :<function> :DELay :USER<n> <delayTime>, (@<channelList>)
[ :SENSe[1]] :<function> :DELay :USER<n> <DEF|MIN|MAX>, (@<channelList>)
[ :SENSe[1]] :<function> :DELay :USER<n>?
[ :SENSe[1]] :<function> :DELay :USER<n>? <DEF|MIN|MAX>
[ :SENSe[1]] :<function> :DELay :USER<n>? (@<channelList>)
[ :SENSe[1]] :<function> :DELay :USER<n>? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The user delay to which this time applies: 1 to 5
<delayTime>	The delay: 0 for no delay, or 167 ns to 10 ks
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

To use this command in a trigger model, assign the delay to the dynamic delay block using the corresponding MEAS<n> parameter that matches the delay number specified here (see the following Example).

The delay is specific to the selected function.

Example

```
:CURRent:DELay:USER1 0.2
:TRIGger:BLOCK:DELay:DYNamic 6, MEAS1
```

Set user delay 1 to 0.2 s for current measurements. Set trigger block 6 to be a dynamic delay that is set to user delay 1 for the function being measured.

Also see

[:TRIGger:BLOCK:DELay:DYNamic](#) (on page 12-292)

[[:SENSe[1]]]:<function>:DETEctor:BANDwidth

This command selects the detector bandwidth for AC current and AC voltage measurements.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	3 (3 Hz)

Usage

```
[[:SENSe[1]]]:<function>:DETEctor:BANDwidth <n>
[[:SENSe[1]]]:<function>:DETEctor:BANDwidth <DEF|MIN|MAX>
[[:SENSe[1]]]:<function>:DETEctor:BANDwidth <n>, (@<channelList>)
[[:SENSe[1]]]:<function>:DETEctor:BANDwidth <DEF|MIN|MAX>, (@<channelList>)
[[:SENSe[1]]]:<function>:DETEctor:BANDwidth?
[[:SENSe[1]]]:<function>:DETEctor:BANDwidth? <DEF|MIN|MAX>
[[:SENSe[1]]]:<function>:DETEctor:BANDwidth? (@<channelList>)
[[:SENSe[1]]]:<function>:DETEctor:BANDwidth? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	3 Hz, 30 Hz, or 300 Hz
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

You can set the detector bandwidth to improve measurement accuracy. Select the bandwidth that contains the lowest frequency component of the input signal. For example, if the lowest frequency component of your input signal is 40 Hz, use a bandwidth setting of 30 Hz.

Example

FUNC "VOLT:AC"	Set the measure function to AC volts.
VOLT:AC:DET:BAND 30	Set the detector bandwidth for AC volts to 30 Hz.

Also see

[\[:SENSe\[1\]\]:<function>:APERture](#) (on page 12-116)
[\[:SENSe\[1\]\]:<function>:AZERo\[:STATe\]](#) (on page 12-133)
[\[:SENSe\[1\]\]:<function>:NPLCycles](#) (on page 12-143)
[\[:SENSe\[1\]\]:AZERo:ONCE](#) (on page 12-177)

[:SENSe[1]] :<function> :INPutimpedance

This command determines when the 10 M Ω input divider is enabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	MOHM10

Usage

```
[ :SENSe[1]] :<function> :INPutimpedance <n>
[ :SENSe[1]] :<function> :INPutimpedance <n>, (@<channelList>)
[ :SENSe[1]] :<function> :INPutimpedance?
[ :SENSe[1]] :<function> :INPutimpedance? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	10 M Ω for all ranges: MOHM10 Automatic: AUTO
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Automatic input impedance provides the lowest measure noise with the highest isolation on the device under test (DUT). When automatic input impedance is selected, the 100 mV to 10 V voltage ranges have more than 10 G Ω input impedance. For the 100 V and 1000 V ranges, a 10 M Ω input divider is placed across the HI and LO input terminals.

When the input impedance is set to 10 M Ω , the 100 mV to 1000 V ranges have a 10 M Ω input divider across the HI and LO input terminals. The 10 M Ω impedance provides stable measurements when the terminals are open (approximately 100 μ V at 1 PLC).

Choosing automatic input impedance is a balance between achieving low DC voltage noise on the 100 mV and 1 V ranges and optimizing measurement noise due to charge injection. The DAQ6510 is optimized for low noise and charge injection when the DUT has less than 100 k Ω input resistance. When the DUT input impedance is more than 100 k Ω , selecting an input impedance of 10 M Ω optimizes the measurement for lowest noise on the 100 mV and 1 V ranges. You can achieve short-term low noise and low charge injection on the 100 mV and 1 V ranges with autozero off. For the 10 V to 1000 V ranges, both input impedance settings achieve low charge injection.

Example

```
:DIG:VOLT:INP AUTO
```

Set input impedance to be set automatically when the digitize voltage function is selected.

Also see

None

[:SENSe[1]]:<function>:LINE:SYNC

This command determines if line synchronization is used during the measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0 (OFF)

Usage

```
[ :SENSe[1]]:<function>:LINE:SYNC <state>
[ :SENSe[1]]:<function>:LINE:SYNC <state>, (@<channelList>)
[ :SENSe[1]]:<function>:LINE:SYNC?
[ :SENSe[1]]:<function>:LINE:SYNC? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<state>	Disable: OFF or 0 Enable: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

When line synchronization is enabled, measurements are initiated at the first positive-going zero crossing of the power line cycle after the trigger.

Example

CURR:LINE:SYNC ON	Turn on line synchronization when DC current is measured.

Also see

[Line cycle synchronization](#) (on page 4-72)

[[:SENSe[1]]]:<function>:NPLCycles

This command sets the time that the input signal is measured for the selected function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
[[:SENSe[1]]]:<function>:NPLCycles <n>
[[:SENSe[1]]]:<function>:NPLCycles <DEF|MIN|MAX>
[[:SENSe[1]]]:<function>:NPLCycles <n>, (@<channelList>)
[[:SENSe[1]]]:<function>:NPLCycles <DEF|MIN|MAX>, (@<channelList>)
[[:SENSe[1]]]:<function>:NPLCycles?
[[:SENSe[1]]]:<function>:NPLCycles? <DEF|MIN|MAX>
[[:SENSe[1]]]:<function>:NPLCycles? (@<channelList>)
[[:SENSe[1]]]:<function>:NPLCycles? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions

<n>	The number of power-line cycles for each measurement: 0.0005 to 15 (60 Hz) or 12 (50 Hz or 400 Hz)
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command sets the amount of time that the input signal is measured.

The amount of time is specified as the number of power line cycles (NPLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz or 400 Hz is 20 ms (1/50). For 60 Hz, if you set the NPLC to 0.1, the measure time is 1.667 ms.

The shortest amount of time results in the fastest reading rate but increases the reading noise and decreases the number of usable digits.

The longest amount of time provides the lowest reading noise and more usable digits but has the slowest reading rate.

Settings between the fastest and slowest number of power line cycles are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits.

NOTE

The measurement time can also be set as an aperture time. Changing the NPLC value changes the aperture time and changing the aperture time changes the NPLC value.

Example 1

```
CURR:NPLC 0.5
```

Sets the measurement time for current measurements to 0.0083 s (0.5/60).

Example 2

```
RES:NPLC 0.5
```

Sets the measurement time for resistance measurements to 0.0083 s (0.5/60).

Example 3

```
VOLT:NPLC 0.5
```

Sets the measurement time for voltage measurements to 0.0083 s (0.5/60).

Example 4

```
VOLT:NPLC 0.5, (@101:105)
```

Sets the measurement time on channels 101 to 105 for voltage measurements to 0.0083 s (0.5/60).

Also see

[\[:SENSe\[1\]\]:<function>:APERture](#) (on page 12-116)

[Using aperture or NPLCs to adjust speed and accuracy](#) (on page 4-73)

[:SENSe[1]] :<function> :OCOMpensated

This command determines if offset compensation is used.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Temperature, 3-wire RTD, 4-wire RTD, or 4-wire resistance: AUTO

Usage

```
[ :SENSe[1]] :<function> :OCOMpensated <state>
[ :SENSe[1]] :<function> :OCOMpensated <state>, (@<channelList>)
[ :SENSe[1]] :<function> :OCOMpensated?
[ :SENSe[1]] :<function> :OCOMpensated? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<state>	Set offset compensation to: - Disabled: OFF - Enabled: ON (for 4-wire resistance, not available for ranges more than 10 kΩ) - Be set automatically for ranges where it is applicable: AUTO
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

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

For 4-wire resistance measurements, when offset compensation is enabled, the measure range is limited to a maximum of 10 kΩ. When Auto is selected, the instrument automatically turns offset compensation on or off as appropriate for the selected range.

For 2-wire resistance measurements, offset compensation is always set to off.

For temperature measurements, offset compensation is only available when the transducer type is set to an RTD option.

Example

<pre>*RST :SENS:FUNC "FRES" :SENS:FRES:RANG 10e3 :FRES:OCOM ON :COUNT 5 :TRAC:TRIG "defbuffer1" :TRAC:DATA? 1, 5, "defbuffer1", READ</pre>	Reset the instrument. Set the measurement function to 4-wire resistance and set the range to 10 kΩ. Turn offset-compensated ohms on. Set the measurement count to 5. Make measurements and store them in defbuffer1. Retrieve the measurement values for readings 1 to 5.

Also see

[Offset-compensated ohms](#) (on page 4-20)

[:SENSe[1]] :<function> :ODETector

This command determines if the detection of open leads is enabled or disabled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	4W Res: OFF (0) Temperature: ON (1)

Usage

```
[ :SENSe[1]] :<function> :ODETector <state>
[ :SENSe[1]] :<function> :ODETector <state>, (@<channelList>)
[ :SENSe[1]] :<function> :ODETector?
[ :SENSe[1]] :<function> :ODETector? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<state>	Disable: OFF or 0 Enable: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURREnt
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

For temperature measurements, this is only available when the transducer is set to a thermocouple or one of the RTDs.

Long lengths of thermocouple wire can have a large amount of capacitance, which is seen at the input of the DMM. If an intermittent open occurs in the thermocouple circuit, the capacitance can cause an erroneous on-scale reading. The open thermocouple detection circuit, when enabled, applies a 100 μ A pulse of current to the thermocouple before the start of each temperature measurement.

Example

TEMP:TRAN TC	Set the transducer type to thermocouple.
TEMP:TC:TYPE K	Set the thermocouple type to K.
TEMP:UNIT CELS	Set the units to Celsius.
TEMP:ODET OFF	Turn open lead detection off.

Also see

None

[[:SENSe[1]]]:<function>:RANGe:AUTO

This command determines if the measurement range is set manually or automatically for the selected measure function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ON (1)

Usage

```
[[:SENSe[1]]]:<function>:RANGe:AUTO <state>
[:SENSe[1]]:<function>:RANGe:AUTO <state>, (@<channelList>)
[:SENSe[1]]:<function>:RANGe:AUTO?
[:SENSe[1]]:<function>:RANGe:AUTO? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions

<state>	Set the measurement range manually: OFF or 0 Set the measurement range automatically: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Autorange selects the best range in which to measure the signal that is applied to the input terminals of the instrument. When autorange is enabled, the range increases at 120 percent of range. The range decreases occur when the reading is <10 percent of nominal range. For example, if you are on the 1 V range and autorange is enabled, the instrument autoranges up to the 10 V range when the measurement exceeds 1.2 V. It autoranges down to the 100 mV range when the measurement falls below 1 V.

This command determines how the range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was last selected by autorange.

When this command is set to on, the instrument automatically goes to the most sensitive range to perform the measurement.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Example

RES:RANG:AUTO ON	Set the range to be selected automatically for resistance measurements.

Also see

[\[:SENSe\[1\]\]:<function>:RANGe\[:UPPer\]](#) (on page 12-149)

[:SENSe[1]]:<function>:RANGe[:UPPer]

This command determines the positive full-scale measure range.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Not applicable

Usage

```
[ :SENSe[1]]:<function>:RANGe[:UPPer] <n>
[ :SENSe[1]]:<function>:RANGe[:UPPer] <DEF|MIN|MAX>
[ :SENSe[1]]:<function>:RANGe[:UPPer] <n>, (@<channelList>)
[ :SENSe[1]]:<function>:RANGe[:UPPer] <DEF|MIN|MAX>, (@<channelList>)
[ :SENSe[1]]:<function>:RANGe[:UPPer]?
[ :SENSe[1]]:<function>:RANGe[:UPPer]? <DEF|MIN|MAX>
[ :SENSe[1]]:<function>:RANGe[:UPPer]? (@<channelList>)
[ :SENSe[1]]:<function>:RANGe[:UPPer]? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions

<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<n>	See Details
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

You can assign any real number using this command. The instrument selects the closest fixed range that is large enough to measure the entered number. For example, for current measurements, if you expect a reading of approximately 9 mA, set the range to 9 mA to select the 10 mA range. When you read this setting, you see the positive full-scale value of the measurement range that the instrument is presently using.

This command is primarily intended to eliminate the time that is required by the instrument to automatically search for a range.

When a range is fixed, any signal greater than the entered range generates an overrange condition. When an overrange condition occurs, the front panel displays "Overflow" and the remote interface returns 9.9e+37.

NOTE

When you set a value for the measurement range, the measurement autorange setting is automatically disabled for the selected measurement function (if supported by that function).

The range for measure functions defaults to autorange for all measure functions. If you switch from a fixed range to autorange, autorange is set to off. The range remains at the fixed range until a measurement is made, at which time the range is set to accommodate the new measurement.

The following table lists the ranges for each function.

If the measurement function is...	The available ranges are...
DC voltage	100 mV, 1 V, 10 V, 100 V, 1000 V
AC voltage	100 mV, 1 V, 10 V, 100 V, 750 V
DC current	10 μ A, 100 μ A, 1 mA, 10 mA, 100 mA, 1 A, 3 A
AC current	100 μ A, 1 mA, 10 mA, 100 mA, 1 A, 3 A
2-wire resistance	10 Ω , 100 Ω , 1 k Ω , 10 k Ω , 100 k Ω , 1 M Ω , 10 M Ω , 100 M Ω
4-wire resistance with offset compensation off	1 Ω , 10 Ω , 100 Ω , 1 k Ω , 10 k Ω , 100 k Ω , 1 M Ω , 10 M Ω , 100 M Ω
4-wire resistance with offset compensation on	1 Ω , 10 Ω , 100 Ω , 1 k Ω , 10 k Ω
Continuity	1 k Ω (fixed)
Diode	10 V (fixed)
Capacitance	1 nF, 10 nF, 100 nF, 1 μ F, 10 μ F, 100 μ F
DC voltage ratio	100 mV, 1 V, 10 V, 100 V, 1000 V

Digitize voltage	100 mV, 1 V, 10 V, 100 V, 1000 V
Digitize current	100 μ A, 1 mA, 10 mA, 100 mA, 1 A, 3 A

Example 1

```
:SENS:CURRE:RANG 10E-6
```

Select the 10 μ A range.

Example 2

```
:DIG:CURRE:RANG 100e-6
```

Select the 100 μ A range.

Example 3

```
:DIG:VOLT:RANG 100e-3
```

Select the 100 mV range.

Also see

[Ranges](#) (on page 4-54)

[\[:SENSef\[1\]\]:<function>:RANGe:AUTO](#) (on page 12-148)

[:SENSe[1]]:<function>:RELative

This command contains the relative offset value.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0

Usage

```
[ :SENSe[1]]:<function>:RELative <n>
[ :SENSe[1]]:<function>:RELative <DEF|MIN|MAX>
[ :SENSe[1]]:<function>:RELative <n>, (@<channelList>)
[ :SENSe[1]]:<function>:RELative <DEF|MIN|MAX>, (@<channelList>)
[ :SENSe[1]]:<function>:RELative?
[ :SENSe[1]]:<function>:RELative? <DEF|MIN|MAX>
[ :SENSe[1]]:<function>:RELative? (@<channelList>)
[ :SENSe[1]]:<function>:RELative? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The relative offset value; see Details
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODe	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command specifies the relative offset value that can be applied to new measurements. When relative offset is enabled, all subsequent measured readings are offset by the value that is set for this command.

You can set this value, or have the instrument acquire a value. If the instrument acquires the value, read this setting to return the value that was measured internally.

The ranges for the relative offset values for all functions are listed in the following table.

	Minimum	Maximum
DC voltage	-1000	1000
AC voltage	-750	750
DC current	-3	3
AC current	-3	3
Resistance	-1e+8	1e+8
4-wire resistance	-1e+8	1e+8
Diode	-10	10
Capacitance	-0.001	0.001
Temperature	-3310	3310
Continuity	-1000	1000
Frequency	-1e+6	1e+6
Period	-1	1
DC voltage ratio - Method set to result	-1e+12	1e+12
DC voltage ratio - Method set to parts	-1000	1000
Digitize voltage	-1000	1000
Digitize current	-3	3

Example

```
CURR:REL 0.5
CURR:REL:STAT ON
```

Set the relative offset for current measurements to 0.5. Enable relative offset.

Also see

[Relative offset](#) (on page 4-56)

[\[:SENSe\[1\]\]:<function>:RELative:ACQuire](#) (on page 12-153)

[\[:SENSe\[1\]\]:<function>:RELative:STATe](#) (on page 12-155)

[:SENSe[1]] :<function> :RELative :ACQuire

This command acquires a measurement and stores it as the relative offset value.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

[:SENSe[1]] :<function> :RELative :ACQuire

<function>	The function to which the setting applies; see Functions

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command triggers the instrument to make a new measurement for the selected function. This measurement is then stored as the new relative offset level.

When you send this command, the instrument does not apply any math, limit test, or filter settings to the measurement, even if they are set. It is a measurement that is made as if these settings are disabled.

You must change to the function for which you want to acquire a value before sending this command. The instrument must have relative offset enabled to use the acquired relative offset value.

After executing this command, you can use the [:SENSe[1]] :<function> :RELative? command to return the last relative level value that was acquired or set.

Example

<pre>FUNC "RES" RES:REL:ACQ RES:REL? RES:REL:STAT ON</pre>	Switch to resistance measurements. Acquire a relative offset value for resistance measurements. Query for the offset value. Turn relative offset on. Example output: -5.4017E-10

Also see

[\[:SENSe\[1\]\] :<function> :RELative](#) (on page 12-151)
[\[:SENSe\[1\]\] :<function> :RELative :STATe](#) (on page 12-155)

[:SENSe[1]] :<function> :RELative :METHod

This command determines if relative offset is applied to the measurements before calculating the DC voltage ratio value.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	PARTs

Usage

```
[ :SENSe[1]] :<function> :RELative :METHod <n>
[ :SENSe[1]] :<function> :RELative :METHod <n>, (@<channelList>)
[ :SENSe[1]] :<function> :RELative :METHod?
[ :SENSe[1]] :<function> :RELative :METHod? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	Apply relative offset: - After calculating the DC voltage ratio value: RESult - Before calculating the DC voltage ratio value: PARTs
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command determines if relative offset is applied to the voltage measurements before the ratio calculation or if the relative offset is applied to the final calculated value.

When the parts method is selected, the individual readings each have the relative offset value applied before being used to calculate the measurement reading. The relative offset value is working with smaller ranges, so an error may occur. Reduce the relative offset value if you receive an error. When a relative offset value is acquired when the parts method is selected, the relative offset levels are made and applied to both input and sense.

A relative offset is applied to the sense value and then to the input value.

When the results method is selected, the individual readings do not have the relative offset value applied. The relative offset value is applied to the final calculation.

Example

```
:FUNC "VOLT:RAT"
:VOLT:RAT:REL:METH PART
```

Set the measure function to DC voltage ratio.
Set the method to apply relative offset before generating the ratio.

Also see

[Relative offset](#) (on page 4-56)

[\[:SENSe\[1\]\]:<function>:RELative:ACQuire](#) (on page 12-153)

[\[:SENSe\[1\]\]:<function>:RELative:STaTe](#) (on page 12-155)

[:SENSe[1]]:<function>:RELative:STaTe

This command enables or disables the application of a relative offset value to the measurement.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	OFF (0)

Usage

```
[ :SENSe[1]]:<function>:RELative:STaTe <state>
[ :SENSe[1]]:<function>:RELative:STaTe <state>, (@<channelList>)
[ :SENSe[1]]:<function>:RELative:STaTe?
[ :SENSe[1]]:<function>:RELative:STaTe? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<state>	Disable the relative offset: OFF or 0 Enable the relative offset: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value. You can enter a relative offset value or have the instrument acquire a relative offset value.

Each returned measured relative reading is the result of the following calculation:

$$\text{Displayed reading} = \text{Actual measured reading} - \text{Relative offset value}$$

Example

```
:SENS:FUNC "VOLT"
:SENS:VOLT:REL 5
:SENSe:VOLT:REL:STaTe ON
```

Set the measurement function to volts with a relative offset of 5 V and enable the relative offset function.

Also see

[Relative offset](#) (on page 4-56)

[\[:SENSe\[1\]\]:<function>:RELative](#) (on page 12-151)

[\[:SENSe\[1\]\]:<function>:RELative:ACQuire](#) (on page 12-153)

[[:SENSe[1]]]:<function>:RTD:ALPHa

This command contains the alpha value of a user-defined RTD.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0.00385055

Usage

```
[[:SENSe[1]]]:<function>:RTD:ALPHa <n>
[:SENSe[1]]:<function>:RTD:ALPHa <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RTD:ALPHa <n>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:ALPHa <DEF|MIN|MAX>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:ALPHa?
[:SENSe[1]]:<function>:RTD:ALPHa? <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RTD:ALPHa? (@<channelList>)
[:SENSe[1]]:<function>:RTD:ALPHa? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	0 to 0.01
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODe	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This attribute is only valid when:

- The function is set to temperature.
- The transducer type is set to one of the RTD options.
- The RTD type is set to user-defined.

Example

<pre>:FUNC "TEMP" :TEMP:TRANsdncer TRTD :TEMP:RTD:THR USER :TEMP:RTD:ALPH 0.00385 :TEMP:RTD:ZERO 120</pre>	Set the measure function to temperature. Set the transducer type to 3-wire RTD. Set the RTD type to User. Set the alpha RTD value to 0.00385. Set the zero RTD value to 120.

Also see

[\[:SENSe\[1\]\]:<function>:RTD:FOUR](#) (on page 12-161)

[\[:SENSe\[1\]\]:<function>:RTD:THRee](#) (on page 12-162)

[\[:SENSe\[1\]\]:<function>:RTD:TWO](#) (on page 12-163)

[\[:SENSe\[1\]\]:<function>:TRANsducer](#) (on page 12-175)

[[:SENSe[1]]]:<function>:RTD:BETA

This command contains the beta value of a user-defined RTD.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	0.10863

Usage

```
[[:SENSe[1]]]:<function>:RTD:BETA <value>
[:SENSe[1]]:<function>:RTD:BETA <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RTD:BETA <value>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:BETA <DEF|MIN|MAX>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:BETA?
[:SENSe[1]]:<function>:RTD:BETA? <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RTD:BETA? (@<channelList>)
[:SENSe[1]]:<function>:RTD:BETA? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<value>	0 to 1
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This attribute is only valid when:

- The function is set to temperature.
- The transducer type is set to one of the RTD options.
- The RTD type is set to user-defined.

Example

<pre>:FUNC "TEMP" :TEMP:TRANsdncer TRTD :TEMP:RTD:THR USER :TEMP:RTD:ALPH 0.005 :TEMP:RTD:DELT 0.00385 :TEMP:RTD:ZERO 120 :TEMP:RTD:BETA 0.3</pre>	Set the measure function to temperature. Set the transducer type to 3-wire RTD. Set the RTD type to User. Set the alpha RTD value to 0.005. Set the delta RTD value to 0.00385. Set the zero RTD value to 120. Set the beta RTD value to 0.3.

Also see

[\[:SENSe\[1\]\]:<function>:RTD:FOUR](#) (on page 12-161)

[\[:SENSe\[1\]\]:<function>:RTD:THRee](#) (on page 12-162)

[\[:SENSe\[1\]\]:<function>:RTD:TWO](#) (on page 12-163)

[\[:SENSe\[1\]\]:<function>:TRANsducer](#) (on page 12-175)

[[:SENSe[1]]]:<function>:RTD:DELtA

This command contains the delta value of a user-defined RTD.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1.4999

Usage

```
[[:SENSe[1]]]:<function>:RTD:DELtA <n>
[:SENSe[1]]:<function>:RTD:DELtA <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RTD:DELtA <n>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:DELtA <DEF|MIN|MAX>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:DELtA?
[:SENSe[1]]:<function>:RTD:DELtA? <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RTD:DELtA? (@<channelList>)
[:SENSe[1]]:<function>:RTD:DELtA? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The delta value from 0 to 5
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This attribute is only valid when:

- The function is set to temperature.
- The transducer type is set to one of the RTD options.
- The RTD type is set to user-defined.

Example

<pre>:FUNC "TEMP" :TEMP:TRANsdncer TRTD :TEMP:RTD:THR USER :TEMP:RTD:ALPH 0.005 :TEMP:RTD:DELT 0.00385 :TEMP:RTD:ZERO 120</pre>	Set the measure function to temperature. Set the transducer type to 3-wire RTD. Set the RTD type to User. Set the alpha RTD value to 0.005. Set the delta RTD value to 0.00385. Set the zero RTD value to 120.

Also see

[\[:SENSe\[1\]\]:<function>:RTD:FOUR](#) (on page 12-161)

[\[:SENSe\[1\]\]:<function>:RTD:THRee](#) (on page 12-162)

[\[:SENSe\[1\]\]:<function>:RTD:TWO](#) (on page 12-163)

[\[:SENSe\[1\]\]:<function>:TRANSducer](#) (on page 12-175)

[:SENSe[1]]:<function>:RTD:FOUR

This command contains the type of 4-wire RTD that is being used.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	PT100

Usage

```
[ :SENSe[1]]:<function>:RTD:FOUR <type>
[ :SENSe[1]]:<function>:RTD:FOUR <type>, (@<channelList>)
[ :SENSe[1]]:<function>:RTD:FOUR?
[ :SENSe[1]]:<function>:RTD:FOUR? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<type>	The type of 4-wire RTD: - PT100: PT100 - PT385: PT385 - PT3916: PT3916 - D100: D100 - F100: F100 - User-specified type: USER
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The transducer type must be set to temperature and the transducer must be set to 4-wire RTD before you can set the RTD type.

Example

```
:FUNC "TEMP"           Set the measure function to temperature.
:TEMP:TRANSducer FRTD   Set the transducer type to 4-wire RTD.
:TEMP:RTD:FOUR PT3916   Set the RTD type to PT3916.
```

Also see

[\[:SENSe\[1\]\]:<function>:TRANSducer](#) (on page 12-175)

[[:SENSe[1]]]:<function>:RTD:THRee

This command defines the type of three-wire RTD that is being used.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	PT100

Usage

```
[[:SENSe[1]]]:<function>:RTD:THRee <type>
[:SENSe[1]]:<function>:RTD:THRee <type>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:THRee?
[:SENSe[1]]:<function>:RTD:THRee? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<type>	The type of three-wire RTD: ■ PT100: PT100 ■ PT385: PT385 ■ PT3916: PT3916 ■ D100: D100 ■ F100: F100 ■ User-specified type: USER
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The transducer type must be set to temperature and the transducer must be set to 3-wire RTD before you can set the RTD type.

Example

```
:FUNC "TEMP"           Set the measure function to temperature.
:TEMP:TRANsdUcer TRTD   Set the transducer type to 3-wire RTD.
:TEMP:RTD:THR PT3916    Set the RTD type to PT3916.
```

Also see

[\[:SENSe\[1\]\]:<function>:TRANsdUcer](#) (on page 12-175)

[Temperature measurements](#) (on page 4-26)

[[:SENSe[1]]]:<function>:RTD:TWO

This command defines the type of 2-wire RTD that is being used.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	PT100

Usage

```
[[:SENSe[1]]]:<function>:RTD:TWO <type>
[:SENSe[1]]:<function>:RTD:TWO <type>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:TWO?
[:SENSe[1]]:<function>:RTD:TWO? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<type>	The type of 2-wire RTD: ■ PT100: PT100 ■ PT385: PT385 ■ PT3916: PT3916 ■ D100: D100 ■ F100: F100 ■ User-specified type: USER
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The transducer type must be set to temperature and the transducer must be set to 2-wire RTD before you can set the RTD type.

Example

<pre>:FUNC "TEMP" :TEMP:TRANsdUcer RTD :TEMP:RTD:TWO PT3916</pre>	Set the measure function to temperature. Set the transducer type to 2-wire RTD. Set the RTD type to PT3916.

Also see

[\[:SENSe\[1\]\]:<function>:TRANsdUcer](#) (on page 12-175)

[Temperature measurements](#) (on page 4-26)

[[:SENSe[1]]]:<function>:RTD:ZERO

This command contains the zero value of a user-defined RTD.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	100

Usage

```
[[:SENSe[1]]]:<function>:RTD:ZERO <n>
[:SENSe[1]]:<function>:RTD:ZERO <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RTD:ZERO <n>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:ZERO <DEF|MIN|MAX>, (@<channelList>)
[:SENSe[1]]:<function>:RTD:ZERO?
[:SENSe[1]]:<function>:RTD:ZERO? <DEF|MIN|MAX>
[:SENSe[1]]:<function>:RTD:ZERO? (@<channelList>)
[:SENSe[1]]:<function>:RTD:ZERO? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The zero value of the RTD: 0 to 10000
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This attribute is only valid when:

- The function is set to temperature.
- The transducer type is set to one of the RTD options.
- The RTD type is set to user-defined.

Example

```
:FUNC "TEMP"
:TEMP:TRANsdncer TRTD
:TEMP:RTD:THR USER
:TEMP:RTD:ALPH 0.00385
:TEMP:RTD:ZERO 120
```

Set the measure function to temperature.
Set the transducer type to 3-wire RTD.
Set the RTD type to User.
Set the alpha RTD value to 0.00385.
Set the zero RTD value to 120.

Also see

[\[:SENSe\[1\]\]:<function>:RTD:THRee](#) (on page 12-162)

[\[:SENSe\[1\]\]:<function>:RTD:TWO](#) (on page 12-163)

[\[:SENSe\[1\]\]:<function>:TRANsducer](#) (on page 12-175)

[:SENSe[1]]:<function>:SRATe

This command defines the precise acquisition rate at which the digitizing measurements are made.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1,000,000

Usage

```
[ :SENSe[1]]:<function>:SRATe <n>
[ :SENSe[1]]:<function>:SRATe <DEF|MIN|MAX>
[ :SENSe[1]]:<function>:SRATe <n>, (@<channelList>)
[ :SENSe[1]]:<function>:SRATe <DEF|MIN|MAX>, (@<channelList>)
[ :SENSe[1]]:<function>:SRATe?
[ :SENSe[1]]:<function>:SRATe? <DEF|MIN|MAX>
[ :SENSe[1]]:<function>:SRATe? (@<channelList>)
[ :SENSe[1]]:<function>:SRATe? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	1,000 to 1,000,000 readings per second
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The sample rate determines how fast the DAQ6510 acquires a digitized reading.

Set the sample rate before setting the aperture. If the aperture setting is too high for the selected sample rate, it is automatically adjusted to the highest aperture that can be used with the sample rate.

Example

```
DIG:FUNC "CURR"
DIG:CURR:SRATE 1000000
DIG:CURR:APER AUTO
DIG:COUN 10
MEAS:DIG?
```

Set the digitize function to measure current. Set the sample rate to 1,000,000, with a count of 10, and automatic aperture. Make a digitize measurement.

Also see

[\[:SENSe\[1\]\]:<function>:APERture](#) (on page 12-116)

[[:SENSe[1]]]:<function>:SENSe:RANGe:AUTO?

This command returns the setting of sense autorange.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	OFF

Usage

[[:SENSe[1]]]:<function>:SENSe:RANGe:AUTO?

<function>	The function to which the setting applies; see Functions
<state>	Autorange disabled: OFF or 0

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This query returns the setting of the sense autorange function.

Example

FUNC "VOLT:RAT"	Set the function to DC voltage ratio.
VOLT:RAT:SENS:RANG:AUTO?	Read the setting of sense autorange.

Also see

[\[:SENSe\[1\]\]:<function>:SENSe:RANGe:UPPer\[?\]](#) (on page 12-167)

[:SENSe[1]] :<function> :SENSe :RANGe [:UPPer] ?

This command displays the positive full-scale range that is being used for the sense measurement.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	10 (10 V)

Usage

```
[ :SENSe[1]] :<function> :SENSe :RANGe [ :UPPer ] ?  
[ :SENSe[1]] :<function> :SENSe :RANGe [ :UPPer ] ? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage [:DC]	RESistance	TEMPerature	VOLTage [:DC] :RATio
VOLTage :AC	FRESistance	CONTinuity	DIGitize :VOLTage
CURRent [:DC]	DIODe	FREQuency [:VOLTage]	DIGitize :CURRent
CURRent :AC	CAPacitance	PERiod [:VOLTage]	

Details

Displays the full-scale input that is used for the reference measurement in the denominator of the ratio. Returns the range in volts.

Example

:SENS:VOLT:RAT:SENS:RANG?	Output the sense range value for the DC voltage ratio function. Example output: 10

Also see

[Ranges](#) (on page 4-54)
[\[:SENSe\[1\]\] :<function> :SENSe :RANGe :AUTO?](#) (on page 12-166)

[:SENSe[1]] :<function> :TCouple :RJUNction :SIMulated

This command sets the simulated reference temperature of the thermocouple reference junction.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Celsius: 23 Kelvin: 296.15 Fahrenheit: 73.4

Usage

```
[ :SENSe[1]] :<function> :TCouple :RJUNction :SIMulated <tempValue>
[ :SENSe[1]] :<function> :TCouple :RJUNction :SIMulated <DEF|MIN|MAX>
[ :SENSe[1]] :<function> :TCouple :RJUNction :SIMulated <tempValue>, (@<channelList>)
[ :SENSe[1]] :<function> :TCouple :RJUNction :SIMulated <DEF|MIN|MAX>, (@<channelList>)
[ :SENSe[1]] :<function> :TCouple :RJUNction :SIMulated?
[ :SENSe[1]] :<function> :TCouple :RJUNction :SIMulated? <DEF|MIN|MAX>
[ :SENSe[1]] :<function> :TCouple :RJUNction :SIMulated? (@<channelList>)
[ :SENSe[1]] :<function> :TCouple :RJUNction :SIMulated? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<tempValue>	The temperature: ■ Celsius: 0 to 65 ■ Kelvin: 273.15 to 338.15 ■ Fahrenheit: 32 to 149
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This attribute applies to the temperature function when the transducer type is set to thermocouple and the reference junction is set to simulated. It allows you to set the simulated reference temperature value.

Example

```
FUNC "TEMP"
TEMP:TRAN TC
TEMP:TC:TYPE K
TEMP:UNIT CELS
TEMP:TC:RJUN:SIM 30
```

Sets 30 °C as the simulated reference temperature for thermocouples.

Also see

[\[:SENSe\[1\]\] :<function> :TCouple :TYPE](#) (on page 12-171)

[\[:SENSe\[1\]\] :<function> :TRANsdncer](#) (on page 12-175)

[\[:SENSe\[1\]\]:<function>:UNIT](#) (on page 12-176)

[Temperature measurements](#) (on page 4-26)

[[:SENSe[1]]]:<function>:TCouple:RJUNction:RSElect

This command selects the type of the thermocouple reference junction.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Simulated

Usage

```
[[:SENSe[1]]]:<function>:TCouple:RJUNction:RSElect <type>
[:SENSe[1]]:<function>:TCouple:RJUNction:RSElect <type>, (@<channelList>)
[:SENSe[1]]:<function>:TCouple:RJUNction:RSElect?
[:SENSe[1]]:<function>:TCouple:RJUNction:RSElect? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<type>	The type of reference junction: ■ SIMulated ■ INTernal ■ EXTernal
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Only available when the temperature function is selected and the transducer type is set to thermocouple.

When you are making rear terminal measurements, you can select the internal or external option. When the internal option is selected, the DAQ6510 periodically updates the temperature. When the external option is selected, the temperature is updated when the external reference function channel is scanned.

When you are making front terminal measurements, the only option is simulated. You can set the simulated reference temperature with the command

```
[[:SENSe[1]]]:<function>:TCouple:RJUNction:SIMulated.
```

Example

TEMP:TRAN TC TEMP:TC:TYPE K TEMP:UNIT CELS TEMP:TC:RJUN:RSEL EXT	Select the external reference junction.

Also see

[\[:SENSe\[1\]\]:<function>:TCouple:RJUNction:SIMulated](#) (on page 12-168)

[\[:SENSe\[1\]\]:<function>:TCouple:TYPE](#) (on page 12-171)

[\[:SENSe\[1\]\]:<function>:TRANsducer](#) (on page 12-175)

[\[:SENSe\[1\]\]:<function>:UNIT](#) (on page 12-176)

[Reference junctions](#) (on page 4-88)

[Temperature measurements](#) (on page 4-26)

[[:SENSe[1]]]:<function>:TCouple:TYPE

This command indicates the thermocouple type.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	K

Usage

```
[[:SENSe[1]]]:<function>:TCouple:TYPE <identifier>
[:SENSe[1]]:<function>:TCouple:TYPE <identifier>, (@<channelList>)
[:SENSe[1]]:<function>:TCouple:TYPE?
[:SENSe[1]]:<function>:TCouple:TYPE? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<identifier>	B, E, J, K, N, R, S, or T
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command is only applicable when the transducer type is set to thermocouple.

Example

```
FUNC "TEMP"
TEMP:TRAN TC
TEMP:TC:TYPE K
TEMP:UNIT CELS
TEMP:TC:RJUN:SIM 30
```

Set the transducer type to thermocouple.
Set the thermocouple type to K.
Set the units to Celsius.
Set the simulated reference temperature to 30.

Also see

[\[:SENSe\[1\]\]:<function>:TCouple:RJUNction:SIMulated](#) (on page 12-168)

[\[:SENSe\[1\]\]:<function>:TRANsducer](#) (on page 12-175)

[Temperature measurements](#) (on page 4-26)

[[:SENSe[1]]]:<function>:THERmistor

This command describes the type of thermistor.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	5000 (5000 Ω)

Usage

```
[[:SENSe[1]]]:<function>:THERmistor <n>
[:SENSe[1]]:<function>:THERmistor <n>, (@<channelList>)
[:SENSe[1]]:<function>:THERmistor?
[:SENSe[1]]:<function>:THERmistor? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The thermistor type in ohms: 2252 Ω: 2252 5000 Ω: 5000 10000 Ω: 10000
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODe	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command is only applicable when the transducer type is set to thermistor.

For the <n> parameter, only 2252, 5000, or 10000 are valid entries. If you enter 2200 or 2250, the DAQ6510 accepts the entry but changes it to 2252. Other values cause an out of range error message.

Example

<pre>FUNC "TEMP" TEMP:TRAN THER TEMP:THER 2252</pre>	Set measurement function to temperature. Set the transducer type to thermistor. Set the thermistor type to 2252.

Also see

[\[:SENSe\[1\]\]:<function>:TRANsducer](#) (on page 12-175)

[Temperature measurements](#) (on page 4-26)

[:SENSe[1]] :<function> :THReshold :RANGe

This command indicates the expected input level of the voltage signal.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	10 (10 V)

Usage

```
[ :SENSe[1]] :<function> :THReshold :RANGe <n>
[ :SENSe[1]] :<function> :THReshold :RANGe <DEF|MIN|MAX>
[ :SENSe[1]] :<function> :THReshold :RANGe <n>, (@<channelList>)
[ :SENSe[1]] :<function> :THReshold :RANGe <DEF|MIN|MAX>, (@<channelList>)
[ :SENSe[1]] :<function> :THReshold :RANGe?
[ :SENSe[1]] :<function> :THReshold :RANGe? <DEF|MIN|MAX>
[ :SENSe[1]] :<function> :THReshold :RANGe? (@<channelList>)
[ :SENSe[1]] :<function> :THReshold :RANGe? <DEF|MIN|MAX>, (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<n>	The range: 0.1 to 750; instrument selects nearest valid range (100 mV, 1 V, 10 V, 100 V, 750 V)
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The range setting conditions the signal. The instrument automatically selects the most sensitive threshold range for the value you enter. For example, if you specify the expected input voltage to be 90 mV, the instrument automatically selects the 100 mV threshold range.

Example

```
FUNC "FREQ"
FREQ:THR:RANG 90
```

Set the threshold range for frequency to 90 V, which will select the 100 V range.

Also see

[\[:SENSe\[1\]\] :<function> :THReshold :RANGe :AUTO](#) (on page 12-174)

[[:SENSe[1]]]:<function>:THReshold:RANGe:AUTO

This command determines if the threshold range is set manually or automatically.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	ON (1)

Usage

```
[[:SENSe[1]]]:<function>:THReshold:RANGe:AUTO <state>
[:SENSe[1]]:<function>:THReshold:RANGe:AUTO <state>, (@<channelList>)
[:SENSe[1]]:<function>:THReshold:RANGe:AUTO?
[:SENSe[1]]:<function>:THReshold:RANGe:AUTO? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<state>	The autorange setting: - Disable: OFF or 0 - Enable: ON or 1
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command determines how the range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was last selected by autorange.

When this command is set to on, the instrument uses the signal to determine the most sensitive range on which to perform the measurement. The instrument sets the range when a measurement is requested. To set the range, the instrument makes a measurement to determine the range before making the final measurement, which can result in slower reading times. Turn autorange off and set a specific range to increase measure time.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Example

```
:FUNC "FREQ"
:FREQ:THR:RANG:AUTO OFF
:FREQ:THR:RANG 10
```

Set measure function to frequency.
Disable the threshold autorange.
Set the range to 10 V.

Also see

[\[:SENSe\[1\]\]:<function>:THReshold:RANGe](#) (on page 12-173)

[[:SENSe[1]]]:<function>:TRANsducer

This command sets the transducer type.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	TCouple

Usage

```
[[:SENSe[1]]]:<function>:TRANsducer <type>
[:SENSe[1]]:<function>:TRANsducer <type>, (@<channelList>)
[:SENSe[1]]:<function>:TRANsducer?
[:SENSe[1]]:<function>:TRANsducer? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<type>	The type of transducer: ■ Thermocouple: TCouple ■ Thermistor: THERmistor ■ 2-wire RTD: RTD ■ 3-wire RTD: TRTD ■ 4-wire RTD: FRTD
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The transducer type determines the type of temperature measurement that is made. Each transducer type has related settings that must also be set. For example, thermocouple measurements are only made if the type is set is set to thermocouple. You also need to set the thermocouple type when setting up a thermocouple. For the RTD transducer types, you also set the RTD type.

Example

<pre>:FUNC "TEMP" :TEMP:TRANsducer FRTD :TEMP:RTD:FOUR PT3916</pre>	Set the measure function to temperature. Set the transducer type to 4-wire RTD. Set the RTD type to PT3916 for 4-wire RTDs.

Also see

[\[:SENSe\[1\]\]:<function>:RTD:FOUR](#) (on page 12-161)

[\[:SENSe\[1\]\]:<function>:RTD:THRee](#) (on page 12-162)

[\[:SENSe\[1\]\]:<function>:RTD:TWO <type>](#) (on page 12-163)

[\[:SENSe\[1\]\]:<function>:TCouple:TYPE](#) (on page 12-171)

[\[:SENSe\[1\]\]:<function>:THERmistor](#) (on page 12-172)

[Temperature measurements](#) (on page 4-26)

[:SENSe[1]]:<function>:UNIT

This command sets the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	Voltage: VOLT Temperature: CELSius

Usage

```
[ :SENSe[1]]:<function>:UNIT <unitOfMeasure>
[ :SENSe[1]]:<function>:UNIT <unitOfMeasure>, (@<channelList>)
[ :SENSe[1]]:<function>:UNIT?
[ :SENSe[1]]:<function>:UNIT? (@<channelList>)
```

<function>	The function to which the setting applies; see Functions
<unitOfMeasure>	Digitize voltage, AC voltage, and DC voltage: VOLT, DB, or DBM Temperature: KELVin, CELSius, or FAHReinheit
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The change in measurement units is displayed when the next measurement is made. You can only change the units for the listed functions.

Example 1

VOLT:UNIT DB	Changes the front-panel display and buffer readings for DC voltage measurements to be displayed in decibels.

Example 2

FUNC "VOLT" VOLT:UNIT DB, (@101:103)	Changes the readings for DC voltage measurements on channels 1 to 3 of slot 1 to be displayed in decibels.

Also see

[Show voltage readings in decibels](#) (on page 4-8)

[Show voltage readings in decibel-milliwatts \(dBm\)](#) (on page 4-8)

[:SENSe[1]]:AZERo:ONCE

This command causes the instrument to refresh the reference and zero measurements once.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
[ :SENSe[1]]:AZERo:ONCE
```

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

This command forces a refresh of the reference and zero measurements that are used for the present aperture setting for the selected function.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

If the NPLC setting is less than 0.2 PLC, sending autozero once can result in delay of more than a second.

Example

FUNC "VOLT" AZER:ONCE	Do a one-time refresh of the reference and zero measurements for the voltage function.

Also see

[Automatic reference measurements](#) (on page 4-53)

[\[:SENSe\[1\]\]:<function>:AZERo\[:STATe\]](#) (on page 12-133)

[[:SENSe[1]]]:CONFIguration:LIST:CATalog?

This command returns the name of one measure configuration list that is stored on the instrument.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

[[:SENSe[1]]]:CONFIguration:LIST:CATalog?

Details

You can use this command to retrieve the names of measure configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns an empty string when there are no more names to return. If the command returns an empty string the first time you send it, no measure configuration lists have been created for the instrument.

Example

CONF:LIST:CAT?	Send this command to retrieve the name of one measure configuration list. To get all stored lists, send it again until it returns an empty string.

Also see

- [Configuration lists](#) (on page 4-99)
- [\[:SENSe\[1\]\]:CONFIguration:LIST:CREate](#) (on page 12-179)

[:SENSe[1]]:CONFIguration:LIST:CREate

This command creates an empty measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
[ :SENSe[1]]:CONFIguration:LIST:CREate "<name>"
```

<name>	A string that represents the name of a measure configuration list

Details

This command creates an empty configuration list. To add configuration indexes to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. To save a configuration list, use a saved setup to store the instrument settings, which include defined configuration lists.

Example

```
:SENS:CONF:LIST:CRE "MyMeasList"
```

Creates a measure configuration list named MyMeasList.

Also see

[*SAV](#) (on page 12-2)

[Configuration lists](#) (on page 4-99)

[\[:SENSe\[1\]\]:CONFIguration:LIST:STORE](#) (on page 12-183)

[:SENSe[1]]:CONFIguration:LIST:DELeTe

This command deletes a measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
[ :SENSe[1]]:CONFIguration:LIST:DELeTe "<name>"
```

```
[ :SENSe[1]]:CONFIguration:LIST:DELeTe "<name>", <index>
```

<name>	A string that represents the name of a measure configuration list
<index>	A number that defines a specific configuration index in the configuration list

Details

Deletes a configuration list. If the index is not specified, the entire configuration list is deleted. If the index is specified, only the specified configuration index in the list is deleted.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and the all the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the higher numbered index first, then the next lower index, and so on. This also means that if you want to delete all the indexes in a configuration list, you must delete index 1 repeatedly until all indexes have been removed.

Example

<code>:SENSe:CONF:LIST:DELeTe "myMeasList"</code>	Deletes a configuration list named myMeasList.
<code>:SENSe:CONF:LIST:DELeTe "myMeasList", 2</code>	Deletes configuration index 2 in a configuration list named myMeasList.

Also see

[Configuration lists](#) (on page 4-99)
[\[:SENSe\[1\]\]:CONFIguration:LIST:CREate](#) (on page 12-179)

[[:SENSe[1]]]:CONFIguration:LIST:QUERy?

This command returns a list of TSP commands and parameter settings that are stored in the specified configuration index.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
[[:SENSe[1]]]:CONFIguration:LIST:QUERy? "<name>", <index>
[:SENSe[1]]:CONFIguration:LIST:QUERy? "<name>", <index>, <fieldSeparator>
```

<name>	A string that represents the name of a measure configuration list
<index>	A number that defines a specific configuration index in the configuration list
<fieldSeparator>	A separator for the data: ▪ Comma (default): COMMa or 1 ▪ Semicolon: SEMicolon or 2 ▪ New line: NEWLine or 3

Details

This command recalls data for one configuration index.

Example

```
:SENS:CONF:LIST:QUER? "MyMeasList", 2, NEWL
```

Returns the TSP commands and parameter settings that represent the settings in configuration index 2.

Example partial output:

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.unit = dmm.UNIT_VOLT
dmm.measure.range = 1
dmm.measure.autorange = dmm.ON
dmm.measure.autozero.enable = dmm.ON
```

Also see

- [Configuration lists](#) (on page 4-99)
- [\[:SENSe\[1\]\]:CONFIguration:LIST:CREate](#) (on page 12-179)
- [*SAV](#) (on page 12-2)
- [TSP command reference](#) (on page 14-1)

[[:SENSe[1]]:CONFIguration:LIST:RECall

This command recalls a configuration index in a measure configuration list.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
[[:SENSe[1]]:CONFIguration:LIST:RECall "<name>"
[:SENSe[1]]:CONFIguration:LIST:RECall "<name>", <index>
```

<name>	A string that represents the name of a measure configuration list
<index>	A number that defines a specific configuration index in the measure configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a measure configuration list. If you do not specify an index when you send the command, it recalls the settings stored in the first configuration index in the specified measure configuration list.

If you recall an invalid index (for example, calling index 3 when there are only two indexes in the configuration list) or try to recall an index from an empty configuration list, event code 2790, "Configuration list error, does not exist" is displayed.

Each index contains the settings for the selected function of that index. Settings for other functions are not affected when the configuration list index is recalled. A single index stores the settings associated with a single measure or digitize function. To see the settings that will be recalled with an index, use the [[:SENSe[1]]:CONFIguration:LIST:QUERy? command.

Example

:SENSe:CONF:LIST:RECall "MyMeasList", 5	Recalls configuration index 5 in a configuration list named MyMeasList.
:SENSe:CONF:LIST:RECall "MyMeasList"	Because an index was not specified, this command recalls configuration index 1 from a configuration list named MyMeasList.

Also see

- [Configuration lists](#) (on page 4-99)
- [\[:SENSe\[1\]\]:CONFIguration:LIST:CREate](#) (on page 12-179)
- [*SAV](#) (on page 12-2)
- [\[:SENSe\[1\]\]:CONFIguration:LIST:QUERy?](#) (on page 12-181)
- [\[:SENSe\[1\]\]:CONFIguration:LIST:STORe](#) (on page 12-183)

[:SENSe[1]]:CONFIguration:LIST:SIZE?

This command returns the size (number of configuration indexes) of a measure configuration list.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
[ :SENSe[1]]:CONFIguration:LIST:SIZE? "<name>"
```

<name>	A string that represents the name of a measure configuration list

Details

This command returns the size (number of configuration indexes) of a measure configuration list. The size of the list is equal to the number of configuration indexes in a configuration list.

Example

```
:SENSe:CONF:LIST:SIZE? "MyMeasList"
```

Returns the number of configuration indexes in a measure configuration list named MyMeasList.
Example output:
3

Also see

[Configuration lists](#) (on page 4-99)

[\[:SENSe\[1\]\]:CONFIguration:LIST:CREate](#) (on page 12-179)

[*SAV](#) (on page 12-2)

[:SENSe[1]]:CONFIguration:LIST:STORe

This command stores the active measure or digitize settings into the named configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Saved settings	Not applicable

Usage

```
[ :SENSe[1]]:CONFIguration:LIST:STORe "<name>"
```

```
[ :SENSe[1]]:CONFIguration:LIST:STORe "<name>", <index>
```

<name>	A string that represents the name of a measure configuration list
<index>	A number that defines a specific configuration index in the configuration list

Details

Use this command to store the active measure or digitize settings to a configuration index in a configuration list. If the index parameter is not provided, the new settings are appended to the end of the list. The index only stores the active settings for a single active measure or digitize function.

A measure configuration list can store measure or digitize settings, but not at the same time. If the active function is a digitize function, digitize settings are saved. When the index is queried, digitize settings and their values are listed, but measure settings are listed as not being used. Similarly, if the active function is a measure function, measure settings are saved. When the index is queried, the measure settings and their values are listed, but the digitize settings are listed as not used.

Configuration lists are not saved when the instrument is turned off or reset. To save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Example

<code>:SENS:CONF:LIST:STOR "MyConfigList"</code>	Stores the active settings of the instrument to the end of the configuration list named MyConfigList.
<code>:SENS:CONF:LIST:STOR "MyConfigList", 5</code>	Stores the active settings of the instrument to the configuration list named MyConfigList in configuration index 5.

Also see

- [Configuration lists](#) (on page 4-99)
- [\[:SENSe\[1\]\]:CONFIguration:LIST:CREate](#) (on page 12-179)
- [*SAV](#) (on page 12-2)

[:SENSe[1]]:COUNT

This command sets the number of measurements to make when a measurement is requested.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
[ :SENSe[1]]:COUNT <n>  
[ :SENSe[1]]:COUNT <DEF|MIN|MAX>  
[ :SENSe[1]]:COUNT <n>, (@<channelList>)  
[ :SENSe[1]]:COUNT <DEF|MIN|MAX>, (@<channelList>)  
[ :SENSe[1]]:COUNT?  
[ :SENSe[1]]:COUNT? <DEF|MIN|MAX>  
[ :SENSe[1]]:COUNT? (@<channelList>)  
[ :SENSe[1]]:COUNT? <DEF|MIN|MAX>, (@<channelList>)
```

<n>	The number of measurements: 1 to 1,000,000 or buffer capacity
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Details

This command sets the number of measurements that are made when a measurement is requested. This command does not affect the trigger model.

This command sets the count for all measure functions.

If you set the count to a value that is larger than the capacity of the reading buffer and the buffer fill mode is set to continuous, the buffer wraps until the number of readings specified have occurred. The earliest readings in the count are overwritten. If the buffer is set to fill once, readings stop when the buffer is filled, even if the count is not complete.

NOTE

To get better performance from the instrument, use the SimpleLoop trigger-model template instead of using the count command.

Example

```
:SENS:FUNC "CURR"  
:TRAC:CLEAR  
:COUN 10  
:MEAS?  
:TRAC:DATA? 1, 10
```

Clear data from the reading buffer.

Set the count to 10.

Make ten measurements.

Returns the last measurement.

Example output:

-5.693831E-05

Read all ten measurements.

Example output:

-7.681046E-05,-2.200288E-04,-9.086048E-05,-6.388056E-05,-7.212282E-05,-4.874761E-05,-4.741654E-04,-6.811028E-05,-5.110232E-05,-5.693831E-05

Also see

[:MEASure?](#) (on page 12-8)

[:TRACe:DATA?](#) (on page 12-237)

[:TRIGger:LOAD "SimpleLoop"](#) (on page 12-330)

[:SENSe[1]]:DIGitize:COUNT

This command sets the number of measurements to digitize when a measurement is requested.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	10,000

Usage

```
[ :SENSe[1]]:DIGitize:COUNT <n>
[ :SENSe[1]]:DIGitize:COUNT <DEF|MIN|MAX>
[ :SENSe[1]]:DIGitize:COUNT <n>, (@<channelList>)
[ :SENSe[1]]:DIGitize:COUNT <DEF|MIN|MAX>, (@<channelList>)
[ :SENSe[1]]:DIGitize:COUNT?
[ :SENSe[1]]:DIGitize:COUNT? <DEF|MIN|MAX>
[ :SENSe[1]]:DIGitize:COUNT? (@<channelList>)
[ :SENSe[1]]:DIGitize:COUNT? <DEF|MIN|MAX>, (@<channelList>)
```

<n>	The number of measurements (1 to 55,000,000)
<DEF MIN MAX>	The DEFault, MINimum, or MAXimum value
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURREnt
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

The digitize function makes the number of readings set by this command in the time set by the sample rate. This command does not affect the trigger model. This command sets the count for all digitize functions.

Example

DIG:FUNC "VOLTage" DIG:COUN 10 MEAS:DIG?	Make ten digitize voltage measurements.

Also see

[:MEASure:DIGitize?](#) (on page 12-6)

[:SENSe[1]]:DIGitize:FUNction[:ON]

This command selects which digitize function is active.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	NONE

Usage

```
[ :SENSe[1]]:DIGitize:FUNction[:ON] "<function>"  
[ :SENSe[1]]:DIGitize:FUNction[:ON] "<function>", (@<channelList>)  
[ :SENSe[1]]:DIGitize:FUNction[:ON]?  
[ :SENSe[1]]:DIGitize:FUNction[:ON]? (@<channelList>)
```

<function>	A string that contains the measurement function to make active: ■ Current: CURREnt ■ Voltage: VOLTage
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Details

Set this command to the type of measurement you want to digitize.

Reading this command returns the digitize function that is presently active.

If you send the query when a measurement function is selected, the query returns `NONE`. The none setting is automatically made if you select a function with `[ :SENSe[1]]:FUNction[:ON]` or through the front panel.

If a channel is closed when you assign a function to the channel, all other channels are opened.

Example

DIG:FUNC "VOLTage"	Make the digitize voltage function the active function.

Also see

[\[:SENSe\[1\]\]:FUNction\[:ON\]](#) (on page 12-189)

[:SENSe[1]]:FUNction[:ON]

This command selects the active measure function.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	VOLT:DC

Usage

```
[ :SENSe[1]]:FUNction[:ON] "<function>"
[ :SENSe[1]]:FUNction[:ON] "<function>", (@<channelList>)
[ :SENSe[1]]:FUNction[:ON]?
[ :SENSe[1]]:FUNction[:ON]? (@<channelList>)
```

<function>	A string that contains the measure function; see Functions
<channelList>	The channels to set, using standard channel naming (on page 11-6)

Functions

VOLTage[:DC]	RESistance	TEMPerature	VOLTage[:DC]:RATio
VOLTage:AC	FRESistance	CONTinuity	DIGitize:VOLTage
CURRent[:DC]	DIODE	FREQuency[:VOLTage]	DIGitize:CURRent
CURRent:AC	CAPacitance	PERiod[:VOLTage]	

Details

Set this command to the type of measurement you want to make.

Reading this command returns the measure function that is presently active.

If you send this query when a digitize measurement function is selected, this returns **NONE**.

If a channel is closed when you assign a function to the channel, all other channels are opened.

Example

:FUNC "VOLTage"	Make the voltage measurement function the active function.

Also see

[\[:SENSe\[1\]\]:DIGitize:FUNction\[:ON\]](#) (on page 12-188)

STATus subsystem

The STATus subsystem controls the status registers of the instrument. For additional information on the status model, see [Status model](#) (on page 16-1).

:STATus:CLEar

This function clears event registers.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:CLEar
```

Details

This command clears the event registers of the Questionable Event and Operation Event Register set. It does not affect the Questionable Event Enable or Operation Event Enable registers.

Example

:STATus:CLEar	Clear the bits in the registers.

Also see

[*CLS](#) (on page 15-2)

:STATus:OPERation:CONDition?

This command reads the Operation Event Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:OPERation:CONDition?
```

Details

This command reads the contents of the Operation Condition Register, which is one of the Operation Event Registers.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-18).

Example

:STAT:OPER:COND?	Returns the contents of the Operation Condition Register.

Also see

[Operation Event Register](#) (on page 16-8)

:STATus:OPERation:ENABLE

This command sets or reads the contents of the Operation Event Enable Register of the status model.

Type	Affected by	Where saved	Default value
Command and query	STATus:PRESet	Not applicable	0

Usage

```
:STATus:OPERation:ENABle <n>  
:STATus:OPERation:ENABle?
```

<n>	The status of the Operation Status Register

Details

This command sets or reads the contents of the Enable register of the Operation Event Register.

When one of these bits is set, when the corresponding bit in the Operation Event Register or Operation Condition Register is set, the OSB bit in the Status Byte Register is set.

When sending binary values, preface <n> with #b. When sending hexadecimal values, preface <n> with #h. No preface is needed when sending decimal values.

Example

```
:STAT:OPER:ENAB #b0101000000000000
```

Sets the 12 and 14 bits of the Operation Status Enable Register using a decimal value.

You could also send the decimal value:

```
:STAT:OPER:ENAB 20480
```

Or the hexadecimal value:

```
:STAT:OPER:ENAB #h5000
```

Also see

[Operation Event Register](#) (on page 16-8)

:STATus:OPERation:MAP

This command allows you to map event numbers to bits in the Operation Event Registers.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:OPERation:MAP <bitNumber>, <setEvent>  
:STATus:OPERation:MAP <bitNumber>, <setEvent>, <clearEvent>  
:STATus:OPERation:MAP? <bitNumber>
```

<bitNumber>	The bit number that is mapped to an event: 0 to 14
<setEvent>	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
<clearEvent>	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

The query requests the mapped set event and mapped clear event status for a bit in the Operation Event Registers. When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

```
:STATus:OPERation:MAP 0, 4917, 4918
```

When event 4917 occurs (a default buffer is 0% filled), bit 0 is set in the condition register and the event register of the Operation Event Register. When event 4918 occurs (a default buffer is 100% filled), bit 0 in the condition register is cleared.

Also see

[Operation Event Register](#) (on page 16-8)

[Programmable status register sets](#) (on page 16-5)

:STATus:OPERation[:EVENT]?

This command reads the Operation Event Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:OPERation[:EVENT]?
```

Details

This attribute reads the operation event register of the status model.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Example

```
STAT:OPER?
```

Returns the contents of the Operation Event Register of the status model.

Also see

[Operation Event Register](#) (on page 16-8)

:STATus:PRESet

This command resets all bits in the status model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:STATus:PRESet

Details

This function clears the event registers and the enable registers for operation and questionable. It will not clear the Service Request Enable Register (*SRE) to Standard Request Enable Register (*ESE).

Preset does not affect the event queue.

The Standard Event Status Register is not affected by this command.

Example

STAT:PRES	Resets the registers.

Also see

[Status model](#) (on page 16-1)

:STATus:QUEStionable:CONDition?

This command reads the Questionable Condition Register of the status model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:STATus:QUEStionable:CONDition?

Details

This command reads the contents of the Questionable Condition Register, which is one of the Questionable Event Registers.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-18).

Example

:STAT:QUES:COND?	Reads the Questionable Condition Register.

Also see

[Questionable Event Register](#) (on page 16-7)

[Understanding bit settings](#) (on page 16-18)

:STATus:QUESTionable:ENABLE

This command sets or reads the contents of the questionable event enable register of the status model.

Type	Affected by	Where saved	Default value
Command and query	STATus:PRESet	Not applicable	0

Usage

```
:STATus:QUESTionable:ENABLE <n>
:STATus:QUESTionable:ENABLE?
```

<n>	The value of the register (0 to 65535)

Details

This command sets or reads the contents of the Enable register of the Questionable Event Register.

When one of these bits is set, when the corresponding bit in the Questionable Event Register or Questionable Condition Register is set, the MSB and QSM bits in the Status Byte Register are set.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-18).

Example

:STAT:QUES:ENAB 8 :STAT:QUES:ENAB?	Enable bit 4, Limit 3 Fail, when the limit test 3 failure value is exceeded. Check to see that the value was set.

Also see

[Questionable Event Register](#) (on page 16-7)

:STATus:QUESTionable:MAP

This command queries mapped event numbers or maps event numbers to bits in the event registers.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	0

Usage

```
:STATus:QUESTionable:MAP <bitNumber>, <setEvent>
:STATus:QUESTionable:MAP <bitNumber>, <setEvent>, <clearEvent>
:STATus:QUESTionable:MAP? <bitNumber>
```

<bitNumber>	The bit number that is mapped to an event: 0 to 14
<setEvent>	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
<clearEvent>	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

```
:STAT:QUES:MAP 0, 4917, 4918
```

When event 4917 occurs (a default buffer is 0% filled), bit 0 is set in the condition register and the event register of the Questionable Event Register. When event 4918 occurs (a default buffer is 100% filled), bit 0 in the condition register is cleared.

Also see

[Questionable Event Register](#) (on page 16-7)

:STATus:QUEStionable[:EVENT]?

This command reads the Questionable Event Register.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:STATus:QUEStionable[:EVENT]?
```

Details

This query reads the contents of the questionable status event register. After sending this command and addressing the instrument to talk, a value is sent to the computer. This value indicates which bits in the appropriate register are set.

The Questionable Register can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set the Questionable Register to the sum of their decimal weights. For example, to set bits B12 and B13, set the Questionable Register to 12,288 (which is the sum of 4,096 + 8,192).

Example

```
:STAT:QUES?
```

Query the Questionable Register.

Also see

[Questionable Event Register](#) (on page 16-7)

SYSTem subsystem

This subsystem contains commands that affect the overall operation of the instrument, such as passwords, beepers, communications, event logs, and time. It also contains queries to determine the cards and channels that are available in the DAQ6510.

:SYSTem:ACcess

This command contains the type of access users have to the instrument through different interfaces.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	FULL

Usage

:SYSTem:ACcess <permissions>
:SYSTem:ACcess?

<permissions>	The level of access that is allowed: ■ Full access for all users from all interfaces: <code>FULL</code> ■ Allows access by one remote interface at a time with login and logout required from other interfaces: <code>EXCLusive</code> ■ Allows access by one remote interface at a time with passwords required on all interfaces: <code>PROTected</code> ■ Allows access by one interface at a time (including the front panel) with passwords required on all interfaces: <code>LOCKout</code>

Details

When access is set to full, the instrument accepts commands from any interface with no login or password.

When access is set to exclusive, you must log out of one remote interface and log into another one to change interfaces. You do not need a password with this access.

Protected access is similar to exclusive access, except that you must enter a password when logging in.

When the access is set to locked out, a password is required to change interfaces, including the front-panel interface.

Under any access type, if a script is running on one remote interface when a command comes in from another remote interface, the command is ignored and the message "FAILURE: A script is running, use ABORT to stop it" is generated.

Example

:SYST:ACC LOCK	Set the instrument access to locked out.
login admin	Log into the interface using the default password.
logout	Log out of the interface.

Also see

[:SYSTem:PASSword:NEW](#) (on page 12-225)

:SYSTem:BEEPer[:IMMediate]

This command generates an audible tone.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:BEEPer[:IMMediate] <frequency>, <duration>

<frequency>	The frequency of the beep (20 Hz to 8000 Hz)
<duration>	The amount of time to play the tone (0.001 s to 100 s)

Details

You can use the beeper of the instrument to provide an audible signal at a specific frequency and time duration.

Using this function from a remote interface does not affect audible errors or key click settings that were made from the DAQ6510 front panel.

Example

:SYSTem:BEEPer 500, 1	Beep at 500 Hz for 1 s.

Also see

None

:SYSTem:CARD<slot>:ACHannel[:START]?

This command indicates the first channel in the specified slot that supports current measurements.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:ACHannel[:START]?

<slot>	The slot number

Details

The channels that support current measurements are grouped, so you can use the start and end channel numbers to identify the group of channels that supports current measurements. If the card supports current measurements, the returned value is the number of the start channel. If only one channel on the card supports current measurements, the start channel matches the end channel. If the channel does not support current measurements, the return is 0.

This command is not available for the 7709 switching module.

Example

syst:card2:ach? syst:card2:ach:end?	If the DAQ6510 contains a 7702 switching module in slot 2, these commands return 41 for the start channel and 42 for the end channel.

Also see

[:SYSTem:CARD<slot>:ACHannel:END?](#) (on page 12-199)

:SYSTem:CARD<slot>:ACHannel:END?

This command indicates the last channel in the specified slot that supports amps measurements.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:ACHannel:END?

<slot>	The slot number

Details

The channels that support amps measurements are grouped, so you can use the start and end channel numbers to identify a group of channels that supports amps measurements. If the card supports amps measurements, the returned value is the number of the end channel. If only one channel on the card supports amps measurements, the end channel matches the start channel. If the channel does not support amps measurements, the return is 0.

This command is not available for the 7709 switching module.

Example

syst:card2:ach? syst:card2:ach:end?	If the DAQ6510 contains a 7702 switching module in slot 2, these commands return 41 for the start channel and 42 for the end channel.

Also see

[:SYSTem:CARD<slot>:ACHannel\[:START\]? \(on page 12-198\)](#)

:SYSTem:CARD<slot>:AOUTput[:STARt]?

This command indicates the first channel in the specified slot that supports analog outputs.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:AOUTput[:STARt]?

<slot>	The slot number

Details

The channels that support analog outputs are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card supports analog outputs, the returned value is the number of the start channel. If only one channel on the card supports analog outputs, the start channel matches the end channel. If the channel does not support analog outputs, the return is 0.

Example

syst:card1:aout? syst:card1:aout:end?	If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 23 for the start channel and 24 for the end channel.

Also see

[:SYSTem:CARD<slot>:AOUTput:END?](#) (on page 12-201)

:SYSTem:CARD<slot>:AOUTput:END?

This command indicates the last channel in the specified slot that supports analog outputs.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:AOUTput:END?

<slot>	The slot number

Details

The channels that support analog outputs are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card supports analog outputs, the returned value is the number of the last channel. If only one channel on the card supports analog outputs, the start channel matches the end channel. If the channel does not support analog outputs, the return is 0.

Example

syst:card1:aout? syst:card1:aout:end?	If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 23 for the start channel and 24 for the end channel.

Also see

[:SYSTem:CARD<slot>:AOUTput\[:STARt\]?](#) (on page 12-200)

:SYSTem:CARD<slot>:CSOhms?

This command indicates whether a card in the specified slot supports commonside channels for 4-wire resistance measurements.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:CSOhms?

<slot>	The slot number

Details

If commonside 4-wire resistance channels are supported, the returned value is 1.

If commonside 4-wire resistance channels are not supported, the return value is 0.

Example

SYST:CARD1:CSO?	Query to verify that slot 1 supports commonside 4-wire resistance channels.

Also see

[:ROUTe\[:CHANnel\]:CSOhms](#) (on page 12-66)

:SYSTem:CARD<slot>:DIGitalio[:STARt]?

This command indicates the first channel in the specified slot that supports digital inputs and outputs.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:DIGitalio[:STARt]?

<slot>	The slot number

Details

The channels that support digital inputs and outputs are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card supports digital inputs and outputs, the returned value is the number of the start channel. If only one channel on the card supports digital inputs and outputs, the start channel matches the end channel. If the channel does not support digital inputs and outputs, the return is 0.

Example

syst:card1:dig? syst:card1:dig:end?	Query if slot 1 supports digital I/O channels. If a 7706 is in slot 1, the return is 21 and 22.

Also see

[:SYSTem:CARD<slot>:DIGitalio:END?](#) (on page 12-204)

:SYSTem:CARD<X>:DIGitalio:END?

This command indicates the last channel in the specified slot that supports digital inputs and outputs.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:DIGitalio:END?

<slot>	The slot number

Details

The channels that support digital inputs and outputs are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card supports digital inputs and outputs, the returned value is the number of the end channel. If only one channel on the card supports digital inputs and outputs, the start channel matches the end channel. If the channel does not support digital inputs and outputs, the return is 0.

Example

syst:card1:dig? syst:card1:dig:end?	If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 21 for the start channel and 22 for the end channel.

Also see

[:SYSTem:CARD<slot>:DIGitalio\[:START\]? \(on page 12-203\)](#)

:SYSTem:CARD<slot>:IDN?

This command returns a string that contains information about the switching module.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:IDN?

<slot>	The slot number: 1 or 2

Details

The information that is returned depends on whether the switching module in the slot is a physical card or pseudocard.

For physical cards, this returns a comma-separated string that contains the model number, description, firmware revision, and serial number of the switching module installed in the specified slot.

For pseudocards, the response is `Pseudo`, followed by the model number, description, and `???` for the firmware revision and serial number.

Example

<pre>syst:card1:ids? syst:card2:ids?</pre>	If a 7706 is installed in slot 1 and a 7700 pseudocard is installed in slot 2, the returns are similar to: <pre>7706,25Ch ALL-IN-ONE w/DIO,01.40e,1243657 7700,Pseudo 20Ch Mux w/CJC,???????,???????????</pre>

Also see

None

:SYSTem:CARD<slot>:ISOLated[:START]?

This command indicates the first channel in the specified slot that has isolated channels.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:ISOLated[:START]?

<slot>	The slot number

Details

The channels that are isolated are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card includes isolated channels, the returned value is the number of the first isolated channel. If only one channel on the module is isolated, the start channel matches the end channel. If none of the channels are isolated, the return is 0.

This command is not available for the 7709 switching module.

Example

syst:card2:isol? syst:card2:isol:end?	If the DAQ6510 contains a 7705 switching module in slot 2, these commands return 1 for the start channel and 40 for the end channel.

Also see

[:SYSTem:CARD<slot>:ISOLated:END?](#) (on page 12-207)

:SYSTem:CARD<slot>:ISOLated:END?

This command indicates the last channel in the specified slot that has isolated channels.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:ISOLated:END?

<slot>	The slot number

Details

The channels that are isolated are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card includes isolated channels, the returned value is the number of the end channel. If only one channel on the card is isolated, the start channel matches the end channel. If none of the channels are isolated, the return is 0.

This command is not available for the 7709 switching module.

Example

<pre>syst:card2:isol? syst:card2:isol:end?</pre>	If the DAQ6510 contains a 7705 switching module in slot 2, these commands return 1 for the start channel and 40 for the end channel.

Also see

[:SYSTem:CARD<slot>:ISOLated\[:START\]?](#) (on page 12-206)

:SYSTem:CARD<slot>:MATRix:COLumns?

This command returns the number of columns in the matrix for the card in the specified slot.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:MATRix:COLumns?

<slot>

The slot number

Details

This attribute is only available if the installed switching module supports matrix channels. If matrix channels are not available, the return value is 0.

Example

SYST: CARD2: MATR: COL?

Returns the number of columns in the matrix on the card in slot 2.
Example output if the switching module is a 7709:
8

Also see

[:SYSTem:CARD<slot>:MATRix:ROWs?](#) (on page 12-208)

:SYSTem:CARD<slot>:MATRix[:ROWs]?

This command returns the number of rows in the matrix on the card in the specified slot.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:MATRix[:ROWs]?

<slot>

The slot number

Details

This attribute is only available if the installed switching module supports matrix channels. If matrix channels are not available, the return value is 0.

Example

SYST: CARD2: MATR?

Returns the number of rows in the matrix on the switching module in slot 2.
Example output:
6

Also see

[:SYSTem:CARD<slot>:MATRix:COLumns?](#) (on page 12-208)

:SYSTem:CARD<slot>:TCHannel[:START]?

This command indicates the first totalizer channel in the specified slot.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:TCHannel[:START]?

<slot>	The slot number

Details

The totalizer channels are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card includes totalizer channels, the returned value is the number of the first totalizer channel. If there is only one totalizer channel, the start channel matches the end channel. If none of the channels are totalizers, the return is 0.

Example

syst:card1:tch? syst:card1:tch:end?	If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 25 for both the start channel and end channel.

Also see

[:SYSTem:CARD<slot>:TCHannel:END?](#) (on page 12-210)

:SYSTem:CARD<slot>:TCHannel:END?

This command indicates the last totalizer channel in the specified slot.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:TCHannel:END?

<slot>	The slot number

Details

The totalizer channels are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card includes totalizer channels, the returned value is the number of the last totalizer channel. If there is only one totalizer channel, the start channel matches the end channel. If none of the channels are totalizers, the return is 0.

Example

syst:card1:tch? syst:card1:tch:end?	If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 25 for both the start channel and end channel.

Also see

[:SYSTem:CARD<slot>:TCHannel:START?](#) (on page 12-209)

:SYSTem:CARD<slot>:TEMPsensor?

This command indicates if the module in the specified slot supports temperature sensor channels.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:TEMPsensor?

<slot>	The slot number

Details

This command is only available for a slot if the installed module supports temperature sensor channels. If temperature sensor channels are not available, the return value is 0.

Example

<pre>syst:card1:temp?</pre>	Query to determine if slot 1 supports temperature sensor channels. If it does support temperature sensor channels, the output is: 1

Also see

[:SYSTem:CARD<slot>:IDN?](#) (on page 12-205)

:SYSTem:CARD<slot>:VCHannel[:START]?

This command indicates the first channel in the specified slot that supports voltage or 2-wire measurements.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:VCHannel[:START]?

<slot>	The slot number

Details

The channels that support voltage or 2-wire measurements are grouped, so you can use the start and end channel numbers to identify the group. If the card supports voltage or 2-wire measurements, the returned value is the number of the start channel. If only one channel on the card supports voltage or 2-wire measurements, the start channel matches the end channel. If the channel does not support voltage or 2-wire measurements, the return is 0.

This command is not available for the 7709 switching module.

Example

syst:card1:vch? syst:card1:vch:end?	These commands return the voltage or 2-wire measurement start channel and end channel.

Also see

[:SYSTem:CARD<slot>:VCHannel:END?](#) (on page 12-213)

:SYSTem:CARD<slot>:VCHannel:END?

This command indicates the last channel in the specified slot that supports voltage or 2-wire measurements.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:VCHannel:END?

<slot>	The slot number

Details

The channels that support voltage or 2-wire measurements are grouped, so you can use the start and end channel numbers to identify the group. If the card supports voltage or 2-wire measurements, the returned value is the number of the last channel. If only one channel on the card supports voltage or 2-wire measurements, the start channel matches the end channel. If the channel does not support voltage or 2-wire measurements, the return is 0.

This command is not available for the 7709 switching module.

Example

syst:card1:vch? syst:card1:vch:end?	These commands return the voltage or 2-wire measurement start channel and end channel.

Also see

[:SYSTem:CARD<slot>:VCHannel\[:START\]? \(on page 12-212\)](#)

:SYSTem:CARD<X>:VMAX?

This command returns the maximum voltage of all channels on a switching module in the specified slot.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CARD<slot>:VMAX?

<X>

The slot number (1 or 2)

Details

This command is only available for a slot if the installed switching module supports voltage settings.

This command is not available for the 7709 switching module.

Example

SYST:CARD1:VMAX?

Example return for a 7706 switching module:
303

Also see

None

:SYSTem:CLEar

This command clears the event log.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:CLEar

Details

This command removes all events from the event log, including entries in the front-panel event log.

Also see

[:SYSTem:ERRor\[:NEXT\]? \(on page 12-218\)](#)

:SYSTem:COMMunication:LAN:CONFigure

This command specifies the LAN configuration for the instrument.

Type	Affected by	Where saved	Default value
Command and query	LXI LAN reset	Nonvolatile memory	AUTO

Usage

```
:SYSTem:COMMunication:LAN:CONFigure "AUTO"  
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPAddress>"  
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPAddress>,<NETmask>"  
:SYSTem:COMMunication:LAN:CONFigure "MANual,<IPAddress>,<NETmask>,<GATeway>"  
:SYSTem:COMMunication:LAN:CONFigure?
```

AUTO	Use automatically configured LAN settings (default)
MANual	Use manually configured LAN settings
<IPAddress>	LAN IP address; must be a string specifying the IP address in dotted decimal notation; required if the mode is set to manual (default "0.0.0.0")
<NETmask>	The LAN subnet mask; must be a string in dotted decimal notation (default "255.255.255.0")
<GATeway>	The LAN default gateway; must be a string in dotted decimal notation (default "0.0.0.0")

Details

This command specifies how the LAN IP address and other LAN settings are assigned. If automatic configuration is selected, the instrument automatically determines the LAN information. When method is automatic, the instrument first attempts to configure the LAN settings using dynamic host configuration protocol (DHCP). If DHCP fails, it tries dynamic link local addressing (DLLA). If DLLA fails, an error occurs.

If manual is selected, you must define the IP address. You can also assign a subnet mask, and default gateway. The IP address, subnet mask, and default gateway must be formatted in four groups of numbers, each separated by a decimal. If you do not specify a subnet mask or default gateway, the previous settings are used. When specifying multiple parameters, do not use spaces after the commas.

The query form of the command returns the present settings in the order shown here.

Automatic:

```
AUTO,<IPAddress>,<NETmask>,<GATeway>
```

Manual:

```
MANual,<IPAddress>,<NETmask>,<GATeway>
```

Example

```
SYST:COMM:LAN:CONF "MANUAL,192.168.0.1,255.255.240.0,192.168.0.3"
SYST:COMM:LAN:CONF?

Set the IP address to be set manually, with the IP address set to 192.168.0.1, the subnet mask to
255.255.240.0, and the gateway address to 192.168.0.3.
Query to verify the settings. The response to the query should be:
manual,192.168.0.1,255.255.240.0,192.168.0.3
```

Also see

[:SYSTem:COMMunication:LAN:MACaddress?](#) (on page 12-217)

:SYSTem:COMMunication:LAN:DST:PROTection

This command allows you to prevent LAN connections from being closed inadvertently.

Type	Affected by	Where saved	Default value
Command and query	Power reset	Not applicable	OFF

Usage

```
SYSTem:COMMunication:LAN:DST:PROTection <state>
SYSTem:COMMunication:LAN:DST:PROTection?
```

<state>	Allow inadvertent closure of LAN connections: OFF or 0 Prevent inadvertent closure of LAN connections: ON or 1

Details

When DST protection is set to off, a simple open-and-close on the DST port (5030) closes any open LAN connections.

When DST protection is set to on, to close any open LAN connection, including the DST port, you must:

1. Open the DST port.
2. Enter the system login and password.
3. Close the DST port.

Setting DST protection to on prevents LAN connections from being inadvertently closed. For example, a LAN connection could be closed if your IT department performs a port scan across the corporate network.

Example

```
SYST:COMM:LAN:PROT ON
SYST:COMM:LAN:CONF?

Set DST protection on.
Query to verify the setting. The response to the query should be:
1
```

Also see

None

:SYSTem:COMMunication:LAN:MACaddress?

This command queries the LAN media access control (MAC) address.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:COMMunication:LAN:MACaddress?
```

Details

The MAC address is a character string representing the MAC address of the instrument in hexadecimal notation. The string includes colons that separate the address octets.

Example

```
:SYSTem:COMMunication:LAN:MACaddress?
```

Returns the MAC address. For example, you might see:
08:00:11:00:00:57

Also see

[:SYSTem:COMMunication:LAN:CONFigure](#) (on page 12-215)

:SYSTem:ERRor[:NEXT]?

This command returns the oldest unread error message from the event log and removes it from the log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:ERRor[:NEXT]?

Details

As error and status messages occur, they are placed in the event log. The event log is a first-in, first-out (FIFO) register that can hold up to 1000 messages.

This command returns the next entry from the event log.

This command does not affect the event log that is displayed on the front panel.

If there are no entries in the event log, the following message is returned:

0,"No error;0;0 0"

This command returns only error messages from the event log. To return information and warning messages, see :SYSTem:EVENTlog:NEXT?.

Note that if you have used :SYSTem:ERRor[:NEXT]? to check events, :SYSTem:EVENTlog:NEXT? shows the next event item after the last error that was returned by :SYSTem:ERRor[:NEXT]?. You will not see warnings or information event log items that occurred before you used :SYSTem:ERRor[:NEXT]?

Example

SYST:ERR:NEXT?	Returns information on the next error in the event log. For example, if you sent a command without a parameter, the return is: -109,"Missing parameter;1;2017/05/06 12:57:04.484"

Also see

[:SYSTem:EVENTlog:NEXT?](#) (on page 12-221)

:SYSTem:ERRor:CODE[:NEXT]?

This command reads the oldest error code.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:ERRor:CODE[:NEXT]?
```

Details

This command returns the numeric code of the next error in the event log. The error is cleared from the queue after being read.

This command returns only error messages from the event log. To return information and warning messages, see :SYSTem:EVENTlog:NEXT?

Example

```
SYST:ERR:CODE?
```

Returns the error code of the next error in the event log. For example, if error -222, Parameter data out of range error, occurred, the output is:
-222

Also see

[:SYSTem:EVENTlog:NEXT?](#) (on page 12-221)

:SYSTem:ERRor:COUNT?

This command returns the number of errors in the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:ERRor:COUNT?
```

Details

This command does not return other types of events, such as information messages. To return other types of events, use :SYSTem:EVENTlog:COUNT?

This command does not clear the errors from the event log.

Example

```
SYST:ERR:COUN?
```

If there are five errors in the event log, the output is:
5

Also see

[:SYSTem:EVENTlog:COUNT?](#) (on page 12-220)

:SYSTem:EVENTlog:COUNT?

This command returns the number of unread events in the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENTlog:COUNT?  
:SYSTem:EVENTlog:COUNT? <eventType>  
:SYSTem:EVENTlog:COUNT? <eventType>, <eventType>  
:SYSTem:EVENTlog:COUNT? <eventType>, <eventType>, <eventType>
```

<eventType>	Limits the list of event log entries to specific types; set to: ▪ Returns the number of errors: ERRor ▪ Returns the number of warnings: WARNing ▪ Returns the number of informational messages: INformational ▪ Returns all events: ALL

Details

A count finds the number of unread events in the event log. You can specify the event types to return or return the count for all events.

This command reports the number of events that have occurred since the command was last sent or since the event log was last cleared.

Example

:SYST:EVEN:COUN? ERR	Displays the present number of errors in the instrument event log. If there are three errors in the event log, output is: 3

Also see

- [:SYSTem:CLEAr](#) (on page 12-214)
- [:SYSTem:EVENTlog:NEXT?](#) (on page 12-221)
- [:SYSTem:EVENTlog:SAVE](#) (on page 12-223)

:SYSTem:EVENTlog:NEXT?

This command returns the oldest unread event message from the event log.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENTlog:NEXT?
:SYSTem:EVENTlog:NEXT? <eventType>
:SYSTem:EVENTlog:NEXT? <eventType>, <eventType>
:SYSTem:EVENTlog:NEXT? <eventType>, <eventType>, <eventType>
```

<eventType>	Limits the event log entries that are returned to specific types; set to: ▪ Returns only the next error: <code>ERRor</code> ▪ Returns only the next warning: <code>WARNing</code> ▪ Returns only the next informational message: <code>INFormational</code> ▪ Returns any event: <code>ALL</code>

Details

When an event occurs on the instrument, it is placed in the event log. The `:SYSTem:EVENTlog:NEXT?` command retrieves an unread event from the event log. Once an event is read, it can no longer be accessed remotely. However, it can be viewed on the front panel.

To read multiple commands, execute this command multiple times.

If there are no entries in the event log, the following is returned:

```
0,"No error;0;0 0"
```

If the event type is not defined, an event of any type is returned.

Note that if you have used `:SYSTem:ERRor[:NEXT]?` to check events, `:SYSTem:EVENTlog:NEXT?` shows the next event item after the last error that was returned by `:SYSTem:ERRor[:NEXT]?`. You will not see warnings or information event log items that occurred before you used `:SYSTem:ERRor[:NEXT]?`.

If the event type is not defined, an event of any type is returned.

The information that is returned is in the order:

```
<eventNumber>, <message>, <eventType>, <timeSeconds>, <timeNanoSeconds>
```

<eventNumber>	The event number
<message>	A description of the event
<eventType>	The type of event: ▪ Error only: 1 ▪ Warning only: 2 ▪ Information only: 4
<timeSeconds>	The seconds portion of the time when the event occurred
<timeNanoSeconds>	The fractional seconds portion of the time when the event occurred

Example

SYST:EVENT:NEXT?	Returns information on the next event in the event log. For example, if you sent a command without a parameter, the return is: -109, "Missing parameter;1;2017/05/06 12:55:33.648"

Also see

- [:SYSTem:CLEAr](#) (on page 12-214)
- [:SYSTem:EVENTlog:SAVE](#) (on page 12-223)

:SYSTem:EVENTlog:POST

This command allows you to post your own text to the event log.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENTlog:POST "<message>"
:SYSTem:EVENTlog:POST "<message>", <eventType>
```

<message>	A string that contains the message that will be associated with this event
<eventType>	The type of event that is generated; set to: - The error type: <code>ERRor</code> - The warning type: <code>WARNing</code> - The informational type: <code>INFormational</code> (default)

Details

You can use this command to create your own event log entries and assign a severity level to them. This can be useful for debugging and status reporting.

From the front panel, you must set the Log Warnings and Log Information options on to have the custom warning and information events placed into the event log.

Example

*CLS SYST:EVENT:POST "my error", INF SYST:EVENT:NEXT?	Clear the event log. Post an error named <code>my error</code> . Output: 1003, "User: my error;4,1400469179,431599191"

Also see

- [Using the event log](#) (on page 3-80)

:SYSTem:EVENTlog:SAVE

This command saves the event log to a file on a USB flash drive.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:SYSTem:EVENTlog:SAVE "<filename>"  
:SYSTem:EVENTlog:SAVE "<filename>", <eventType>
```

<filename>	A string that holds the name of the file to be saved
<eventType>	Limits the event log entries that are saved to specific types; set to: ■ ERRor: Saves only error entries ■ WARNing: Saves only warning entries ■ INFormational: Saves only informational entries ■ ALL: Saves all event log entries (default)

Details

This command saves all event log entries to a USB flash drive.

If you do not define an event type, the instrument saves all event log entries.

The extension `.csv` is automatically added to the file name.

Example

```
SYST:EVEN:SAVE "/usb1/July_error_log", ERR
```

Saves the error events in the event log to a file on the USB flash drive named `July_error_log.csv`.

Also see

[:SYSTem:CLEAr](#) (on page 12-214)

:SYSTem:GPIB:ADDRess

This command contains the GPIB address.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	16

Usage

```
:SYSTem:GPIB:ADDRess <n>  
:SYSTem:GPIB:ADDRess?
```

<n>	The GPIB address of the instrument: 1 to 30

Details

The address can be set to any address value from 1 to 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

A new GPIB address takes effect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so use care when setting this attribute from the GPIB interface.

You should allow sufficient time for the command to be processed before attempting to communicate with the instrument again.

*RST does not affect the GPIB address.

Example

:SYSTem:GPIB:ADDReSS 26 :SYSTem:GPIB:ADDReSS?	Sets the GPIB address and reads the address. Output: 2.600000e+01

Also see

[GPIB setup](#) (on page 2-10)

:SYSTem:LFRequency?

This query contains the power line frequency setting that is used for NPLC calculations.

Type	Affected by	Where saved	Default value
Query only	Power cycle	Not applicable	Not applicable

Usage

:SYSTem:LFRequency?

Details

The instrument automatically detects the power line frequency when the instrument is powered on. Power line frequency can be 50 Hz, 60 Hz, or 400 Hz. If the line frequency is 400 Hz, 50 Hz is returned.

Example

:SYST:LFR?	Check the line frequency.

Also see

None

:SYSTem:PASSword:NEW

This command stores the instrument password.

Type	Affected by	Where saved	Default value
Command only	Rear-panel LAN reset	Nonvolatile memory	admin

Usage

:SYSTem:PASSword:NEW "<password>"

<password>	A string that contains the instrument password (maximum 30 characters)

Details

When the access to the instrument is set to protected or lockout, this is the password that is used to gain access.

If you forget the password, you can reset the password to the default:

1. On the front panel, press **MENU**.
2. Under System, select **Info/Manage**.
3. Select **Password Reset**.

Example

SYST:PASS:NEW "N3wpa55w0rd"	Change the password of the instrument to N3wpa55w0rd.

Also see

[:SYSTem:ACCess](#) (on page 12-196)

:SYSTem:PCARd<slot>

This command specifies a pseudocard to implement for the designated slot.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	0

Usage

:SYSTem:PCARd<slot> <cardNumber>

<slot>	Slot number (1 or 2)
<cardNumber>	The card number: 0: No pseudocard 7700: 20-channel Differential Multiplexer Module 7701: 32-channel Differential Multiplexer Module 7702: 40-channel Differential Multiplexer Module with Screw Terminals 7703: 32-channel High Speed Differential Multiplexer Module 7705: 40-channel Single-pole Control Module 7706: 20-channel All-in-One I/O Module 7707: 32-channel Digital I/O Module with 10-channel Differential Multiplexer 7708: 40-channel Differential Multiplexer Module with Automatic CJC and Screw Terminals 7709: 6 × 8 Matrix Module 7710: 20-channel Solid-state Differential Multiplexer with Automatic CJC 7711: 2 GHz 50 Ohm RF Module 7712: 3.5 GHz 50 Ohm RF Module

Details

Pseudocards allow you to configure your system without having an actual switching module installed in your system. You can perform open, close, and scan operations and configure your system with pseudocards.

This command is only applicable to a slot that does not have a switching module or pseudocard installed. If a pseudocard is presently assigned to the slot, you must set the slot to no pseudocard before assigning the new pseudocard.

After assigning a pseudocard, you can use valid commands for the switching module for that slot.

Changing the pseudocard assignment from a pseudocard to no pseudocard invalidates scan lists that include that slot.

Example

<pre>syst:pcar2 0 syst:pcar2 7700 syst:card2:idsn?</pre>	Remove any existing pseudocards from slot 2. Install a pseudocard 7700 in slot 2 and verify that it is installed. A typical return is: 7700,Pseudo 20Ch Mux w/CJC,???????,???????????

Also see

- [Pseudocards](#) (on page 5-2)
- [:SYSTem:CARD<slot>:IDN?](#) (on page 12-205)

:SYSTem:POSetup

This command selects the defaults when you power on the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	RST

Usage

<pre>:SYSTem:POSetup <name> :SYSTem:POSetup?</pre>	
<name>	Which setup to restore when you power on the instrument: ■ Power on to *RST defaults: RST ■ Stored setup 0: SAV0 ■ Stored setup 1: SAV1 ■ Stored setup 2: SAV2 ■ Stored setup 3: SAV3 ■ Stored setup 4: SAV4

Details

- When you select RST, the instrument restores settings to their default values when the instrument is powered on.
- When you select a SAV option, the settings in the selected saved setup are applied when the instrument is powered on. The settings are saved using the *SAV command.

Example

<pre>SYST:POS SAV1</pre>	Set the instrument to restore the settings that are saved in the stored setup 1 when the instrument is powered on.

Also see

- [*SAV](#) (on page 12-2)

:SYSTem:TIME

This command sets the absolute time of the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	See Details

Usage

```
:SYSTem:TIME <year>, <month>, <day>, <hour>, <minute>, <second>
:SYSTem:TIME <hour>, <minute>, <second>
:SYSTem:TIME?
:SYSTem:TIME? 1
```

<year>	Year; must be more than 1970
<month>	Month (1 to 12)
<day>	Day (1 to 31)
<hour>	Hour in 24-hour time format (0 to 23)
<minute>	Minute (0 to 59)
<second>	Second (0 to 59)

Details

When queried without a parameter, this command returns the present timestamp value in seconds since January 1, 1970 to the nearest second.

If you query with 1, this command returns the present timestamp in the format:

```
<weekday> <month> <day> <hour>:<minute>:<second> <year>
```

Where <weekday> is the day of the week.

Internally, the instrument bases time in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

Example

<pre>syst:time 2018, 2, 15, 11, 30, 30 syst:time? 1</pre>	Set the system time to February 15, 2018 at 11:30:30 and confirm setting. Output: Thu Feb 15 11:30:35 2018

Also see

None

:SYSTem:VERSion?

This command returns the present SCPI version.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:SYSTem:VERSion?

Details

This query command returns the SCPI version.

Example

SYSTem:VERSion?	Query the version. An example of a return is: 1996.0

Also see

None

TRACe subsystem

The TRACe subsystem contains commands that control the reading buffers.

:TRACe:ACTual?

This command contains the number of readings in the specified reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:TRACe:ACTual?
:TRACe:ACTual? "<bufferName>"

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

This command returns the number of readings stored in the buffer.

Example

TRACe:MAKE "testData", 200 COUN 10 MEASure:CURRent? "testData"	Creates 200 element reading buffer named <code>testData</code> . Set the measurement count to 10. Set the measurement function to current. Make readings and store the readings in <code>testData</code> . Returns the tenth measurement reading after taking all ten readings.
:TRACe:ACTual?	Returns the number of readings in <code>defbuffer1</code> . Example output: 850
:TRACe:ACTual? "testData"	Returns the number of readings in the buffer <code>testData</code> . Example output: 10

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)

:TRACe:ACTual:END?

This command indicates the last index in a reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:TRACe:ACTual:END?
:TRACe:ACTual:END? "<bufferName>"

<bufferName>	A string that indicates the reading buffer; the default buffers (<code>defbuffer1</code> or <code>defbuffer2</code>) or the name of a user-defined buffer; if no buffer is specified, <code>defbuffer1</code> is used

Details

Use this command to find the ending index in a reading buffer.

Example

```
TRACe:MAKE "test1", 100
COUNT 6
MEASure:CURRent? "test1"
:TRACe:ACTual:STARt? "test1" ; END? "test1"
MEASure:CURRent? "test1"
:TRACe:ACTual:STARt? "test1" ; END? "test1"

Create a buffer named test1 with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer test1.
Get the start index and end index of test1.
Output: 1;6
Make six more measurements and store them in buffer test1.
Get the start and end index of test1.
Output: 1;12
```

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:ACTual:STARt?](#) (on page 12-231)
- [:TRACe:MAKE](#) (on page 12-243)

:TRACe:ACTual:STARt?

This command indicates the starting index in a reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:ACTual:STARt?
:TRACe:ACTual:STARt? "<bufferName>"
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

Use this command to find the starting index in a reading buffer.

Example

```
TRACe:MAKE "test1", 100  
COUNT 6  
MEASure:CURRent? "test1"  
:TRACe:ACTual:STArT? "test1" ; END? "test1"
```

Create a buffer named `test1` with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer `test1`.
Get the start index and end index of `test1`.
Output: 1;6

Also see

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

[:TRACe:ACTual:END?](#) (on page 12-230)

[:TRACe:MAKE](#) (on page 12-243)

:TRACe:CHANnel:MATH

This command sets a math expression on a channel.

Type	Affected by	Where saved	Default value
Command only	Instrument reset Power cycle	Not saved	NONE

Usage

```
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, NONE
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, ADD,
    (@<channelNum2>)
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, AVERage,
    (@<channelNum2>)
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, DIVide,
    (@<channelNum2>)
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, EXPonent
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, LOG
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, MULTiply,
    (@<channelNum2>)
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, POLY, <constant0>,
    <constant1>, <constant2>, <constant3>, <constant4>, <constant5>
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, POWer, <constant0>
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, RATE
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, RECiprocal
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, SQRoot
:TRACe:CHANnel:MATH "<bufferName>", (@<channelNumber>), <units>, SUBtract,
    (@<channelNum2>)
```

<bufferName>	String that contains the name of the reading buffer; must be set to the style <code>FULL</code>	
<channelNumber>	String that contains the channel to apply the expression to, using normal channel list syntax; this is also the channel that supplies the <i>x</i> value in the expressions	
<units>	The units to be applied to the value generated by the expression:	
	■ DC current: AMP ■ AC current: AMPAC ■ Celsius: CELSius ■ Custom unit 1: CUSTOM1 ■ Custom unit 2: CUSTOM2 ■ Custom unit 3: CUSTOM3 ■ DAC (voltage): DAC ■ Decibel-milliwatts: DBM ■ Decibels: DECibel ■ Digital I/O: DIO ■ Fahrenheit: FAHRenheit ■ Capacitance: FARad	■ Frequency: HERTz ■ Kelvin: KELVin ■ No unit: NONE ■ Resistance: OHM ■ Percent: PERCent ■ DC voltage ratio: RATio ■ Reciprocal: RECiprocal ■ Period: SECond ■ Totalizer: TOT ■ DC voltage: VOLT ■ AC voltage: VOLTAC ■ Power: WATT
<channelNum2>	String that contains the channel from which to get the previous measurement (the <i>a</i> value in the expressions); a measurement must be made on this channel before the <i>channelNumber</i>	

<constant0>	The constant to be used for c0 in the expression
<constant1>	The constant to be used for c1 in the expression
<constant2>	The constant to be used for c2 in the expression
<constant3>	The constant to be used for c3 in the expression
<constant4>	The constant to be used for c4 in the expression
<constant5>	The constant to be used for c5 in the expression

Details

This command applies a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You can apply expressions to readings made from the DMM inputs or readings made from channels. If you have expressions set through both the buffer math and channel math commands, the expressions set for the channel math command take precedence.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style `FULL`. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

The expressions you can apply to readings are listed in the following table. In the formulas:

- `r` = present reading
- `a` = previous reading
- `t` = timestamp of the reading
- `c` = constant

Expression	<expression>	Formula
No math applied	NONE	Not applicable
Add	ADD	$r + a$
Average	AVERage	$\frac{(r+a)}{2}$
Divide	DIVide	$\frac{r}{a}$
Exponent	EXPonent	10^r
Log10	LOG10	$\log_{10} r$
Multiply	MULTiply	$r * a$

Polynomial	POLY	$c0 + c1 \cdot r + c2 \cdot r^2 + c3 \cdot r^3 + c4 \cdot r^4 + c5 \cdot r^5$
Power	POWer	r^c
Rate of change	RATE	$\frac{(r-r_1)}{(t-t_1)}$
Reciprocal	RECiprocal	$\frac{1}{r}$
Square root	SQRoot	$\sqrt{r}$
Subtract	SUBtract	$r - a$

Example

```
*RST
TRAC:MAKE "expressions", 100, FULL

ROUT:SCAN (@101:109)
ROUT:SCAN:BUFF "expressions"
SENS:FUNC "VOLT", (@101:109)

TRACe:CHANnel:MATH "expressions", (@102), AMP, ADD, (@101)
INIT
*WAI

TRAC:DATA? 1, 9, "expressions", READ, EXTR
DISP:SCR READ
```

This example assumes a 7700 switching module in slot 1.

Instrument has terminals set to REAR.

Reset the instrument.

Make a buffer named `expressions`, set to store 100 readings with a style of `FULL`.

Set up a scan that includes channels 1 to 10.

Set the scan to use the reading buffer `expressions`.

Set the function for the scan channels to voltage.

Set up channel math for channel 2, using a unit of A, that adds channels 1 and 2.

Start the scan.

Wait for results.

Read the data from channels 1 to 9, including the readings and the value generated by the expression.

Display the reading table on the front panel of the instrument.

Also see

[:TRACe:MATH](#) (on page 12-245)

[:TRACe:UNIT](#) (on page 12-262)

:TRACe:CLEar

This command clears all readings and statistics from the specified buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:CLEar
:TRACe:CLEar "<bufferName>"
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Example

TRACe:MAKE "testData", 200 MEASure:RESistance? "testData" TRACe:ACTual? "testData" TRACe:CLEar "testData" TRACe:ACTual? "testData"	Create user-defined buffer named testData. Make a measurement and store it in testData and return the last reading measured. Verify that there is data in testData buffer. Output: 1 Clear testData buffer. Verify that testData is empty. Output: 0
TRACe:CLEar TRACe:CLEar "defbuffer1" TRACe:CLEar "defbuffer2"	Clear the default buffer. This command clears defbuffer1. Clear defbuffer1. Specify default buffer by name. Clear defbuffer2. Specify default buffer by name.

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)

:TRACe:DATA?

This command returns specified data elements from a specified reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:DATA? <startIndex>, <endIndex>
:TRACe:DATA? <startIndex>, <endIndex>, "<bufferName>"
:TRACe:DATA? <startIndex>, <endIndex>, "<bufferName>", <bufferElements>
```

<startIndex>	Beginning index of the buffer to return; must be 1 or greater
<endIndex>	Ending index of the buffer to return
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<bufferElements>	A list of elements in the buffer to print; if nothing is specified, READing is used; see Details for the list of options for buffer elements; 14 comma-delimited buffer elements may be specified

Details

The output of :TRACe:DATA? is affected by the data format selected by :FORMat[:DATA]. If you set FORMat[:DATA] to REAL or SREAL, you will have fewer options for buffer elements. The only buffer elements available are READing, RELative, and EXTRa. If you request a buffer element that is not permitted for the selected data format, the instrument generates the error 1133, "Parameter 4, Syntax error, expected valid name parameters."

NOTE

To change the number of digits returned in a remote command reading, use the :FORMat:AScii:PRECision command.

When specifying buffer elements, you can:

- Specify buffer elements in any order.
- Include up to 14 elements in a single list. You can repeat elements as long as the number of elements in the list is less than 14.

Use a comma to delineate multiple elements for a data point.

The options for <bufferElements> are described in the following table.

Option	Description
CHANnel	The channel from which the data was acquired
DATE	The date when the data point was measured; the buffer style must be set to the style standard or full to use this option
EXTRa	Returns an additional value (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAFORMatted	Returns the measurement and the unit of measure of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
EXTRAUNIT	Returns the units of additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option
FORMatted	The measured value as it appears on the front panel
FRACTIONal	The fractional seconds when the data point was measured
READing	The measurement reading
RELative	The relative time when the data point was measured
SECONDS	The seconds in UTC (Coordinated Universal Time) format when the data point was measured
STATus	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table below
TIME	The time when the data point was measured
TSTamp	The timestamp when the data point was measured
UNIT	The unit of measure of the measurement

The `STATus` buffer element returns status values for the readings in the buffer. The status values are integers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description
0x0001	STAT_QUESTIONABLE	1	Measure status questionable
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize)
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high
0x0100	STAT_START_GROUP	256	First reading in a group

Example

```
TRAC:MAKE "buf100", 100
TRIGger:LOAD "SimpleLoop", 5, 0, "buf100"
INIT
*WAI
TRAC:DATA? 1, 5, "buf100", READ, REL
TRAC:DATA? 1, 5, "buf100", REL
TRAC:DATA? 1, 3, "buf100"
```

Create a buffer called `buf100` with a maximum size of 100.

Set the instrument to configure the trigger model to loop, taking five readings with no delay, and store the readings in the `buf100` reading buffer.

Initiate the trigger model and wait for the trigger model to complete. The trigger model makes five readings and stores them in `buf100`.

Read five data points and include the reading and relative time for each data point.

Output:

```
5.043029E-05,0.000000,5.016920E-05,0.020199,5.047250E-05,0.040201,5.001598E-05,
0.079671,5.053504E-05,0.099205
```

Read five data points and include relative time for each data point.

Output:

```
0,0.020199,0.040201,0.079671,0.099205
```

Returns the first three reading values from `buf100` reading buffer.

Output:

```
5.043029E-05,5.016920E-05,5.047250E-05
```

Also see

[:FORMat\[:DATA\]](#) (on page 12-58)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

[:TRACe:MAKE](#) (on page 12-243)

:TRACe:DELeTe

This command deletes a user-defined reading buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:DELeTe "<bufferName>"
```

<bufferName>	A string that contains the name of the user-defined reading buffer to delete

Details

You cannot delete the default reading buffers, defbuffer1 and defbuffer2.

Example

TRAC:DEL "testData"	Delete the testData buffer.

Also see

[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-30)
[:TRACe:MAKE](#) (on page 12-243)

:TRACe:FILL:MODE

This command determines if a reading buffer is filled continuously or is filled once and stops.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1: CONT defbuffer2: CONT User-defined buffers: ONCE

Usage

```

:TRACe:FILL:MODE <fillType>
:TRACe:FILL:MODE <fillType>, "<bufferName>"
:TRACe:FILL:MODE?
:TRACe:FILL:MODE? "<bufferName>"

```

<fillType>	Fill the buffer continuously: CONTinuous Fill the buffer, then stop: ONCE
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

When a reading buffer is set to fill once, no data is overwritten in the buffer. When the buffer is filled, no more data is stored in that buffer and new readings are discarded.

When a reading buffer is set to fill continuously, the oldest data is overwritten by the newest data after the buffer fills.

When you change the fill mode of a buffer, any data in the buffer is cleared.

Example

TRACe:MAKE "testData", 100	Create a user-defined reading buffer named <code>testData</code> with a capacity of 100 measurements.
TRACe:FILL:MODE? "testData"	Query the fill mode setting for <code>testData</code> .
TRACe:FILL:MODE CONT, "testData"	Output: ONCE
TRACe:FILL:MODE? "testData"	Set <code>testData</code> fill mode to continuous.
	Query the fill mode setting for <code>testData</code> .
	Output: CONT
TRACe:FILL:MODE?	Query the fill mode setting for <code>defbuffer1</code> .
	Output: CONT

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)
- [:TRACe:CLEAr](#) (on page 12-236)

:TRACe:LOG:STATe

This command indicates if information events are logged when the specified reading buffer is at 0% or 100% filled.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	defbuffer1: ON (1) defbuffer2: ON (1) User-created buffer: OFF (0)

Usage

```
:TRACe:LOG:STATe <logState>
:TRACe:LOG:STATe <logState>, "<bufferName>"
:TRACe:LOG:STATe?
:TRACe:LOG:STATe? "<bufferName>"
```

<logState>	Do not log information events: OFF or 0 Log information events: ON or 1
<bufferName>	A string that indicates the reading buffer; the default buffers (<code>defbuffer1</code> or <code>defbuffer2</code>) or the name of a user-defined buffer; if no buffer is specified, <code>defbuffer1</code> is used

Details

If this is set to on, when the reading buffer is cleared (0% filled) or full (100% filled), an event is logged in the event log. If this is set to off, reading buffer status is not reported in the event log.

Example

```
TRACe:LOG:STATe?
```

Query the log state of defbuffer1.

Output:

1

Indicates that the log state is on.

Also see

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

[:TRACe:MAKE](#) (on page 12-243)

[Using the event log](#) (on page 3-80)

:TRACe:MAKE

This command creates a user-defined reading buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRACe:MAKE "<bufferName>", <bufferSize>  
:TRACe:MAKE "<bufferName>", <bufferSize>, <bufferStyle>
```

<bufferName>	A user-supplied string that indicates the name of the buffer
<bufferSize>	A number that indicates the maximum number of readings that can be stored in <bufferName>; minimum is 10; set to 0 to maximize the buffer size (see Details)
<bufferStyle>	The type of reading buffer to create: - Store readings with full accuracy with formatting: <code>STANdard</code> (default) - Store the same information as standard, plus additional information, such as the ratio component of a DCV ratio measurement: <code>FULL</code> - Store external reading buffer data: <code>WRITable</code> - Store external reading buffer data with two reading values: <code>FULLWRITable</code>

Details

The buffer name for a user-defined reading buffer cannot be `defbuffer1` or `defbuffer2`. In addition, the buffer name must not already exist as a global variable, a local variable, table, or array.

If you create a reading buffer that has the same name as an existing user-defined buffer, the event message 1115, "Parameter error: TRACe:MAKE cannot take an existing reading buffer name" is generated.

When you create a reading buffer, it becomes the active buffer. If you create two reading buffers, the last one you create becomes the active buffer.

If you select 0, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The default fill mode of a user-defined buffer is once. You can change it to continuous later.

Once the buffer style is selected, it cannot be changed.

Not all remote commands are compatible with the writable and full writable buffer styles. Check the Details section of the command descriptions before using them with any of these buffer styles.

Writable reading buffers are used to bring external data into the instrument. You cannot assign them to collect data from the instrument.

You can change the buffer capacity for an existing buffer through the front panel or by using the `:TRACe:POINtS` command.

Example 1

<code>TRACe:MAKE "capTrace", 200, WRITable</code>	Create a 200-element writable reading buffer named <code>capTrace</code> .

Example 2

```
TRACe:MAKE "bufferVolts", 100
TRACe:POINts? "bufferVolts"

TRACe:DELeTe "bufferVolts"
TRACe:MAKE "bufferVolts", 1000
TRACe:POINts?
```

Create a buffer named `bufferVolts` to store 100 readings.
Query the size of `bufferVolts`.
Output:
100
Delete the buffer named `bufferVolts`.
Make a new buffer named `bufferVolts` to store 1000 readings.
Query the size of `bufferVolts` again to verify it can store 1000 readings.
Output:
1000

Example 3

```
TRACe:POINts 5000, "bufferVolts"
TRACe:POINts?
```

Resize an existing buffer named `bufferVolts` to store 5000 readings.
Query the size of `bufferVolts` to verify it can store 5000 readings.
Output:
5000

Also see

[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-30)
[:TRACe:FILL:MODE](#) (on page 12-240)
[:TRACe:POINts](#) (on page 12-248)
[:TRACe:WRITe:FORMat](#) (on page 12-264)
[:TRACe:WRITe:READIng](#) (on page 12-266)

:TRACe:MATH

This command allows you to run a mathematical expression on a measurement. The expression is applied when the measurement is placed in the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Instrument reset Power cycle	Not saved	NONE

Usage

```
:TRACe:MATH "<bufferName>", <units>, ADD
:TRACe:MATH "<bufferName>", <units>, AVERage
:TRACe:MATH "<bufferName>", <units>, DIVide
:TRACe:MATH "<bufferName>", <units>, EXPonent
:TRACe:MATH "<bufferName>", <units>, LOG10
:TRACe:MATH "<bufferName>", <units>, MULTiply
:TRACe:MATH "<bufferName>", <units>, NONE
:TRACe:MATH "<bufferName>", <units>, POLY, <constant0>, <constant1>, <constant2>,
    <constant3>, <constant4>, <constant5>
:TRACe:MATH "<bufferName>", <units>, POWer, <constant0>
:TRACe:MATH "<bufferName>", <units>, RATE
:TRACe:MATH "<bufferName>", <units>, RECiprocal
:TRACe:MATH "<bufferName>", <units>, SQRoot
:TRACe:MATH "<bufferName>", <units>, SUBtract
```

<bufferName>	String that contains the name of the reading buffer; must be set to the style FULL	
<units>	The units to be applied to the value generated by the expression:	
	■ DC current: AMP ■ AC current: AMPAC ■ Celsius: CELSius ■ Custom unit 1: CUSTOM1 ■ Custom unit 2: CUSTOM2 ■ Custom unit 3: CUSTOM3 ■ DAC (voltage): DAC ■ Decibel-milliwatts: DBM ■ Decibels: DECibel ■ Digital I/O: DIO ■ Fahrenheit: FAHRenheit ■ Capacitance: FARad	■ Frequency: HERTz ■ Kelvin: KELVIn ■ No unit: NONE ■ Resistance: OHM ■ Percent: PERCent ■ DC voltage ratio: RATio ■ Reciprocal: RECiprocal ■ Period: SECond ■ Totalizer: TOT ■ DC voltage: VOLT ■ AC voltage: VOLTAC ■ Power: WATT
<constant0>	The constant to be used for c0 in the expression	
<constant1>	The constant to be used for c1 in the expression	
<constant2>	The constant to be used for c2 in the expression	
<constant3>	The constant to be used for c3 in the expression	
<constant4>	The constant to be used for c4 in the expression	
<constant5>	The constant to be used for c5 in the expression	

Details

This command applies a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You can apply expressions to readings made from the DMM inputs or readings made from channels. If you have expressions set through both the buffer math and channel math commands, the expressions set for the channel math command take precedence.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style `FULL`. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

The expressions you can apply to readings are listed in the following table. In the formulas:

- `r` = present reading
- `a` = previous reading
- `t` = timestamp of the reading
- `c` = constant

Expression	<expression>	Formula
No math applied	NONE	Not applicable
Add	ADD	$r + a$
Average	AVERage	$\frac{(r+a)}{2}$
Divide	DIVide	$\frac{r}{a}$
Exponent	EXPonent	10^r
Log10	LOG10	$\log_{10} r$
Multiply	MULTiply	$r * a$
Polynomial	POLY	$c0 + c1 \cdot r + c2 \cdot r^2 + c3 \cdot r^3 + c4 \cdot r^4 + c5 \cdot r^5$
Power	POWER	r^c
Rate of change	RATE	$\frac{(r-r_1)}{(t-t_1)}$
Reciprocal	RECiprocal	$\frac{1}{r}$

Square Root	SQRoot	$\sqrt{r}$
Subtract	SUBtract	$r - a$

Example

```
*RST
TRAC:MAKE "expressions", 100, FULL
SENS:FUNC "VOLT"
TRACe:MATH "expressions", VOLT, ADD
COUN 10
READ? "expressions"
TRAC:DATA? 1, 10, "expressions", READ, EXTR
DISP:SCR READ
```

Instrument has terminals set to FRONT.

Reset the instrument.

Make a buffer named `expressions`, set to store 100 readings with a style of FULL.

Set the measure function to voltage.

Set up buffer math, using a unit of V, that adds the present and previous readings.

Make a reading and store it in the `expressions` buffer.

Read the data in buffer indexes 1 to 10, including the readings and the values generated by the expression.

Display the reading table on the front panel of the instrument.

Also see

[:TRACe:CHANnel:MATH](#) (on page 12-233)

[:TRACe:UNIT](#) (on page 12-262)

:TRACe:POINts

This command sets the number of readings a buffer can store.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRACe:POINts <newSize>  
:TRACe:POINts <newSize>, "<bufferName>"  
:TRACe:POINts?  
:TRACe:POINts? "<bufferName>"
```

<newSize>	The new size for the buffer; set to 0 to maximize the buffer size (see Details)
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

This command allows you to change or view how many readings a buffer can store. Changing the size of a buffer will cause any existing data in the buffer to be lost.

If you select 0, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The overall capacity of all buffers stored in the instrument can be up to 6,000,000 readings for standard reading buffers.

For more information about buffer capacity, see [Setting reading buffer capacity](#) (on page 6-11).

Example

TRACe:MAKE "testData", 100 TRACe:POINts 300, "testData" TRACe:POINts? "testData" TRACe:POINts?	Create a user-defined reading buffer named <code>testData</code> with a capacity of 100 measurements. Change the buffer capacity to 300. Query the capacity of <code>testData</code> . Output: 300 Query the capacity of the default buffer. Output: 10000

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)

:TRACe:SAVE

This command saves data from the specified reading buffer to a USB flash drive.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:SAVE "<fileName>"
:TRACe:SAVE "<fileName>", "<bufferName>"
:TRACe:SAVE "<fileName>", "<bufferName>", <what>
:TRACe:SAVE "<fileName>", "<bufferName>", <what>, <start>, <end>
```

<fileName>	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<what>	Defines which information is saved in the file on the USB flash drive: ■ All information: ALL ■ Dates, times, and fractional seconds are saved; the default value: FORMat ■ Relative timestamps are saved: RELative ■ Seconds and fractional seconds are saved: RAW ■ Timestamps are saved: STAMp ■ Standard set of data: STANdard ■ Brief set of data (reading and relative timestamp only): BRIEF ■ Extra data, such as the reading and unit from a DC voltage ratio measurement: EXTRa ■ Channel information: CHANCOL ■ Compact information formatted so that it is easy to graph in Microsoft Excel: EASYGRAPH If nothing is defined, the output defaults to the reading, unit, range digits, display digits, status, extra values, and channel
<start>	Defines the starting point in the buffer to start saving data
<end>	Defines the ending point in the buffer to stop saving data

Details

The file name must specify the full path (including /usb1/). If included, the file extension must be set to .csv. If no file extension is specified, .csv is added.

The DAQ6510 does not check for existing files when you save. Verify that you are using a unique name to avoid overwriting any existing .csv files on the flash drive.

Example

```
TRACe:MAKE "MyBuffer", 100
SENSe:COUNT 5
MEASure:CURRent:DC? "MyBuffer"
TRACe:DATA? 1, 5, "MyBuffer", READ, REL
TRACe:SAVE "/usb1/myData.csv", "MyBuffer"
TRACe:SAVE "/usb1/myDataRel.csv", "MyBuffer", REL
```

Create a buffer called `MyBuffer` with a maximum size of 100.

Make five readings for each measurement request and return the data.

Make the measurements.

Read the reading and relative timestamp value for each point from 1 to 5.

Output:

```
-0.000000,0.000000,
  -0.000000,
  0.301759,-0.000000,0.579068,-0.000000,
  0.884302,-0.000000,1.157444
```

Save all reading and default time information from a buffer named `MyBuffer` to a file named `myData.csv` on the USB flash drive.

Save all readings and relative timestamps from `MyBuffer` to a file named `myDataRel.csv` on the USB flash drive.

Also see

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

[:TRACe:MAKE](#) (on page 12-243)

[:TRACe:SAVE:APPend](#) (on page 12-251)

:TRACe:SAVE:APPend

This command appends data from the reading buffer to a file on the USB flash drive.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:SAVE:APPend "<fileName>"
:TRACe:SAVE:APPend "<fileName>", "<bufferName>"
:TRACe:SAVE:APPend "<fileName>", "<bufferName>", <timeFormat>
:TRACe:SAVE:APPend "<fileName>", "<bufferName>", <timeFormat>, <start>, <end>
```

<fileName>	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<timeFormat>	Indicates how date and time information from the buffer is saved in the file on the USB flash drive; the values are: ■ All information: ALL ■ Dates, times, and fractional seconds are saved; the default value: FORMat ■ Relative timestamps are saved: RELative ■ Seconds and fractional seconds are saved: RAW ■ Timestamps are saved: STAMP ■ Standard set of data: STANdard ■ Brief set of data (reading and relative timestamp only): BRIEF ■ Extra data, such as the reading and unit from a DC voltage ratio measurement: EXTRa ■ Channel information: CHANCOL ■ Compact information formatted so that it is easy to graph in Microsoft Excel: EASYGRAPH If nothing is defined, the output defaults to the reading, unit, range digits, display digits, status, extra values, and channel
<start>	Defines the starting point in the buffer to start saving data
<end>	Defines the ending point in the buffer to stop saving data

Details

If the file you specify does not exist on the USB flash drive, this command creates the file.

For options that save more than one item of time information, each item is comma-delimited. For example, the default format is date, time, and fractional seconds for each reading.

The file extension `.csv` is appended to the file name if necessary. Any file extension other than `.csv` generates an error.

The index column entry in the `.csv` file starts at 1 for each append operation.

Example

```
TRACe:MAKE "testData", 100
SENSe:COUNT 5
MEASure:CURRENT:DC? "testData", READ, REL
TRACe:SAVE "/usb1/myData5.csv", "testData"
TRACe:CLEAr
MEASure:CURRENT:DC?
TRACe:SAVE:APPend "/usb1/myData5.csv", "defbuffer1"
MEASure:CURRENT:DC? "testData"
TRACe:SAVE:APPend "/usb1/myData5.csv", "testData", RAW, 6, 10
```

Create a buffer called `testData`.

Make 5 readings and return the fifth point, which will contain the reading and relative timestamp value. Store the buffer data in the `myData5.csv` file.

Clear `defbuffer1`.

Make 5 readings, store them in `defbuffer1`, and return the fifth reading.

Append all the readings stored in `defbuffer1` to the `myData5.csv` file.

Take 5 more readings, store them in `testData`, and return the fifth reading.

Append all the readings stored in positions 6 through 10 `testData` to the `myData5.csv` file using raw timestamps.

Also see

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

[:TRACe:MAKE](#) (on page 12-243)

:TRACe:STATistics:AVERage?

This command returns the average of all readings in the buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:AVERage?  
:TRACe:STATistics:AVERage? "<bufferName>"  
:TRACe:STATistics:AVERage? "<bufferName>", (@<channelName>)
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<channelName>	The channel for which to return an average

Details

This command returns the average reading calculated from all the readings in the specified reading buffer.

When the reading buffer is configured to fill continuously and overwrite old data with new data, the buffer statistics include the data that was overwritten. To get statistics that do not include data that has been overwritten, define a large buffer size that will accommodate the number of readings you will make.

Example

TRACe:STAT:AVERage?	Returns the average reading for the readings in the default buffer defbuffer1.
TRACe:STAT:AVERage? "testData"	Returns the average reading for the readings in the user-defined buffer testData.

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)
- [:TRACe:STATistics:CLEAr](#) (on page 12-254)
- [:TRACe:STATistics:MAXimum?](#) (on page 12-255)
- [:TRACe:STATistics:MINimum?](#) (on page 12-256)
- [:TRACe:STATistics:PK2Pk?](#) (on page 12-257)
- [:TRACe:STATistics:STDDev?](#) (on page 12-259)

:TRACe:STATistics:CLEar

This command clears the statistical information associated with the specified buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:CLEar
:TRACe:STATistics:CLEar "<bufferName>"
```

<bufferName>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; if no buffer is defined, clears the statistics from defbuffer1

Details

This command clears the statistics without clearing the readings.

Example

TRACe:STATistics:CLEar	Clear all statistics in defbuffer1.
TRACe:STATistics:CLEar "testData"	Clears all statistics in a user-defined buffer named testData.

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)
- [:TRACe:STATistics:AVERage?](#) (on page 12-253)
- [:TRACe:STATistics:MAXimum?](#) (on page 12-255)
- [:TRACe:STATistics:MINimum?](#) (on page 12-256)
- [:TRACe:STATistics:PK2Pk?](#) (on page 12-257)
- [:TRACe:STATistics:STDDev?](#) (on page 12-259)

:TRACe:STATistics:MAXimum?

This command returns the maximum reading value in the reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:MAXimum?  
:TRACe:STATistics:MAXimum? "<bufferName>"  
:TRACe:STATistics:MAXimum? "<bufferName>", (@<channelName>)
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<channelName>	The channel for which to return a maximum

Example

TRACe:STAT:MAXimum?	Returns the maximum reading value in the default buffer, defbuffer1.
TRACe:STAT:MAXimum? "testData"	Returns the maximum reading value in the user-defined buffer testData.

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)
- [:TRACe:STATistics:AVERage?](#) (on page 12-253)
- [:TRACe:STATistics:CLEar](#) (on page 12-254)
- [:TRACe:STATistics:MINimum?](#) (on page 12-256)
- [:TRACe:STATistics:PK2Pk?](#) (on page 12-257)
- [:TRACe:STATistics:STDDev?](#) (on page 12-259)

:TRACe:STATistics:MINimum?

This command returns the minimum reading value in the reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:MINimum?  
:TRACe:STATistics:MINimum? "<bufferName>"  
:TRACe:STATistics:MINimum? "<bufferName>", (@<channelName>)
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<channelName>	The channel for which to return a minimum

Example

TRACe:STAT:MINimum?	Returns the minimum reading value in the default buffer defbuffer1.
TRACe:STAT:MINimum? "testData"	Returns the minimum reading value in the user-defined buffer testData.

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)
- [:TRACe:STATistics:AVERage?](#) (on page 12-253)
- [:TRACe:STATistics:CLEar](#) (on page 12-254)
- [:TRACe:STATistics:MAXimum?](#) (on page 12-255)
- [:TRACe:STATistics:PK2Pk?](#) (on page 12-257)
- [:TRACe:STATistics:STDDev?](#) (on page 12-259)

:TRACe:STATistics:PK2Pk?

This command returns the peak-to-peak value of all readings in the reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:PK2Pk?  
:TRACe:STATistics:PK2Pk? "<bufferName>"  
:TRACe:STATistics:PK2Pk? "<bufferName>", (@<channelName>)
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<channelName>	The channel for which to return the peak-to-peak value

Example

TRACe:STAT:PK2Pk?	Returns the peak-to-peak reading value in the default buffer defbuffer1.
TRACe:STAT:PK2Pk? "testData"	Returns the peak-to-peak reading value in the user-defined buffer testData.

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)
- [:TRACe:STATistics:AVERAge?](#) (on page 12-253)
- [:TRACe:STATistics:CLEAr](#) (on page 12-254)
- [:TRACe:STATistics:MAXimum?](#) (on page 12-255)
- [:TRACe:STATistics:MINimum?](#) (on page 12-256)
- [:TRACe:STATistics:STDDev?](#) (on page 12-259)

:TRACe:STATistics:SPAN?

This command contains the number of readings in the specified reading buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:SPAN?  
:TRACe:STATistics:SPAN? "<bufferName>"  
:TRACe:STATistics:SPAN? "<bufferName>", (@<channelName>)
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<channelName>	The channel for which to return the number of readings

Example

TRACe:STAT:SPAN?	Returns the number of readings in default buffer defbuffer1.
TRACe:STAT:SPAN? "testData"	Returns the number of readings in the user-defined buffer testData.

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)
- [:TRACe:STATistics:AVERage?](#) (on page 12-253)
- [:TRACe:STATistics:CLEar](#) (on page 12-254)
- [:TRACe:STATistics:MAXimum?](#) (on page 12-255)
- [:TRACe:STATistics:MINimum?](#) (on page 12-256)
- [:TRACe:STATistics:PK2Pk?](#) (on page 12-257)
- [:TRACe:STATistics:STDDev?](#) (on page 12-259)

:TRACe:STATistics:STDDev?

This command returns the standard deviation of all readings in the buffer.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:STATistics:STDDev?  
:TRACe:STATistics:STDDev? "<bufferName>"  
:TRACe:STATistics:STDDev? "<bufferName>", (@<channelName>)
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used
<channelName>	Channels for which to show the standard deviation

Example

TRACe:STAT:STDDev?	Returns the standard deviation of the readings in the default buffer defbuffer1.
TRACe:STAT:STDDev? "testData"	Returns the standard deviation of the readings in the user-defined buffer testData.

Also see

- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [:TRACe:MAKE](#) (on page 12-243)
- [:TRACe:STATistics:CLEAr](#) (on page 12-254)
- [:TRACe:STATistics:MAXimum?](#) (on page 12-255)
- [:TRACe:STATistics:MINimum?](#) (on page 12-256)
- [:TRACe:STATistics:PK2Pk?](#) (on page 12-257)

:TRACe:TRIGger

This command makes readings using the active measure function and stores them in a reading buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:TRIGger
:TRACe:TRIGger "<bufferName>"
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

A measure function must be selected before sending this command.

This command makes the number of measurements that is set by the count command.

Example

TRACe:MAKE "MyBuffer", 100 COUN 5 TRACe:TRIG "MyBuffer" TRACe:DATA? 1, 5, "MyBuffer", rel	Create a buffer called MyBuffer with a maximum size of 100. Make readings and store them in MyBuffer. Recall the relative time when the data points were measured for the first five readings in the buffer. Example output: 0.000000,0.408402,0.816757,1.208823,1.617529

Also see

- [\[:SENSe\[1\]\]:COUNT](#) (on page 12-185)
- [\[:SENSe\[1\]\]:FUNCTION\[:ON\]](#) (on page 12-189)
- [:TRACe:DATA?](#) (on page 12-237)
- [:TRACe:MAKE](#) (on page 12-243)

:TRACe:TRIGger:DIGitize

This command makes readings using the active digitize function and stores them in the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRACe:TRIGger:DIGitize
:TRACe:TRIGger:DIGitize "<bufferName>"
```

<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

A digitize function must be selected before sending this command.

This command makes the number of digitize measurements that is set by the digitize count command.

Example

DIG:FUNC "VOLTage" TRACe:MAKE "MyBuffer", 60000 TRACe:TRIG:DIG "MyBuffer" TRACe:TRIG:DIG "MyBuffer" TRACe:TRIG:DIG "MyBuffer" TRACe:TRIG:DIG "MyBuffer" TRACe:TRIG:DIG "MyBuffer" TRACe:DATA? 1, 5, "MyBuffer", rel	Make the digitize voltage measurement function the active function. Create a buffer called <code>MyBuffer</code> with a maximum size of 60000. Make readings and store them in <code>MyBuffer</code> . Recall the relative time when the data points were measured for the first five readings in the buffer. Example output: 0.000000,0.408402,0.816757,1.208823,1.617529

Also see

- [\[:SENSe\[1\]\]:DIGitize:COUNt](#) (on page 12-187)
- [\[:SENSe\[1\]\]:DIGitize:FUNCTION\[:ON\]](#) (on page 12-188)
- [:TRACe:MAKE](#) (on page 12-243)

:TRACe:UNIT

This command allows you to create up to three custom units of measure for use in buffers.

Type	Affected by	Where saved	Default value
Command only	Power cycle	Not saved	CUSTOM1: X CUSTOM2: Y CUSTOM3: Z

Usage

:TRACe:UNIT CUSTOM<n>, "<unitOfMeasure>"

<n>	The number of the custom unit, 1, 2, or 3
<unitOfMeasure>	A string that defines the custom unit; up to three characters

Details

You can use custom units of measures in buffer math, channel math, and writable buffers.

If you specify more than two characters, the additional characters are ignored. Some characters are converted to other symbols:

- u is displayed as μ .
- dC is displayed as $^{\circ}\text{C}$.
- dF is displayed as $^{\circ}\text{F}$.
- RA is displayed as V/V .

This unit is reset when power is cycled. It is not affected by reset.

Example

```
*RST
TRAC:MAKE "expressions", 100, FULL
SENS:FUNC "VOLT"
TRAC:UNIT CUSTOM1, "fb"
TRAC:MATH "expressions", CUSTOM1, ADD
COUN 10
READ? "expressions"
TRAC:DATA? 1, 10, "expressions", READ, EXTR
DISP:SCR READ
```

Instrument has terminals set to FRONT.

Reset the instrument.

Make a buffer named `expressions`, set to store 100 readings with a style of FULL.

Set the measure function to voltage.

Set the custom 1 buffer unit to `fb`.

Set up buffer math, using the custom 1 unit of measure, that adds the present and previous readings.

Set the instrument to make 10 measurements.

Make a reading and store it in the `expressions` buffer.

Read the data in buffer indexes 1 to 10, including the readings and the values generated by the expression.

Display the reading table on the front panel of the instrument.

Also see

[:TRACe:CHANnel:MATH](#) (on page 12-233)

[:TRACe:MATH](#) (on page 12-245)

[:TRACe:WRITe:FORMat](#) (on page 12-264)

:TRACe:WRITe:FORMat

This command sets the units and number of digits of the readings that are written into the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRACe:WRITe:FORMat "<bufferName>", <units>, <displayDigits>
:TRACe:WRITe:FORMat "<bufferName>", <units>, <displayDigits>, <extraUnits>
:TRACe:WRITe:FORMat "<bufferName>", <units>, <displayDigits>, <extraUnits>,
    <extraDigits>
```

<bufferName>	A user-supplied string that indicates the name of the buffer	
<units>	The units for the first measurement in the buffer index:	
	■ AMP ■ AMP_AC ■ AMPAC ■ CELSius ■ CUSTOM1 (user-defined unit) ■ CUSTOM2 (user-defined unit) ■ CUSTOM3 (user-defined unit) ■ DAC ■ DBM ■ DECibel ■ DIO ■ FAHRenheit ■ FARad ■ HERTz	■ KELVin ■ NONE ■ OHM ■ PERCent ■ RATio ■ RECiprocal ■ SECond ■ TOT ■ VOLT ■ VOLT_AC ■ VOLTAC ■ WATT ■ X
<displayDigits>	The number of digits to use for the first value in the buffer index: 3 to 8	
<extraUnits>	The units for the second measurement in the buffer index; the selections are the same as <units>; if this parameter is not specified, the value for <units> is used; extra units are only valid for buffer style FULLWRITable	
<extraDigits>	The number of digits to use for the second measurement; the selections are the same as <displayDigits>; if this parameter is not specified, the value for <displayDigits> is used; extra digits are only valid for buffer style FULLWRITable	

Details

This command is valid when the buffer style is writable or full writable. When the buffer style is set to full writable, you can include an extra value.

The format defines the units and the number of digits that are reported for the data. This command affects how the data is shown in the reading buffer and what is shown on the front-panel Home, Histogram, Reading Table, and Graph screens.

Example 1

```
:TRAC:MAKE "write2me", 1000, WRITable
:TRAC:WRIT:FORM "write2me", WATT, 4
:TRAC:WRIT:READ "write2me", 1
:TRAC:WRIT:READ "write2me", 2
:TRAC:WRIT:READ "write2me", 3
:TRAC:WRIT:READ "write2me", 4
:TRAC:WRIT:READ "write2me", 5
:TRAC:WRIT:READ "write2me", 6
:TRAC:DATA? 1, 6, "write2me", read, unit
```

Creates a 1000-point reading buffer named `write2me`. Style is writable.

Set the data format to show units of watts with 4-½ digit resolution.

Write six pieces of data into the buffer.

Read the buffer.

Output:

```
1.000000E+00,Watt DC,2.000000E+00,Watt DC,3.000000E+00,Watt DC,4.000000E+00,Watt
DC,5.000000E+00,Watt DC,6.000000E+00,Watt DC
```

Example 2

```
:TRAC:MAKE "write2me", 1000, FULLWRIT
:TRAC:WRIT:FORM "write2me", WATT, 4, WATT, 4
:TRAC:WRIT:READ "write2me", 1, 7
:TRAC:WRIT:READ "write2me", 2, 8
:TRAC:WRIT:READ "write2me", 3, 9
:TRAC:WRIT:READ "write2me", 4, 10
:TRAC:WRIT:READ "write2me", 5, 11
:TRAC:WRIT:READ "write2me", 6, 12
:TRAC:DATA? 1, 6, "write2me", read, unit, read, unit
```

Creates a 1000-point reading buffer named `write2me`. Style is full writable.

Set the data format to show units of watts with 4½ digit resolution for the first value and the second value in the buffer index.

Write 12 pieces of data into the buffer.

Read the buffer.

Output:

```
1.000000E+00,Watt DC,7.000000E+00,Watt DC,2.000000E+00,Watt DC,8.000000E+00,Watt
DC,3.000000E+00,Watt DC,9.000000E+00,Watt DC,4.000000E+00,Watt
DC,1.000000E+01,Watt DC,5.000000E+00,Watt DC,1.100000E+01,Watt
DC,6.000000E+00,Watt DC,1.200000E+01,Watt DC
```

Also see

[Reading buffers](#) (on page 6-1)

[:TRACe:MAKE](#) (on page 12-243)

[:TRACe:WRITe:READIng](#) (on page 12-266)

[Writable reading buffers](#) (on page 6-36)

:TRACe:WRITE:READING

This command allows you to write readings into the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

For buffers that are set to the writable buffer style:

```
:TRACe:WRITE:READING "<bufferName>", <readingValue>
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <seconds>
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <seconds>, <fractionalSeconds>
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <seconds>,
    <fractionalSeconds>, <status>
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <seconds>,
    <fractionalSeconds>, <status>, <channel>
```

For buffers that are set to the full writable buffer style:

```
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <extraValue>
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <extraValue>, <seconds>
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <extraValue>, <seconds>,
    <fractionalSeconds>
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <extraValue>, <seconds>,
    <fractionalSeconds>, <status>
:TRACe:WRITE:READING "<bufferName>", <readingValue>, <extraValue>, <seconds>,
    <fractionalSeconds>, <status>, <channel>
```

<bufferName>	A user-supplied string that indicates the name of the buffer
<readingValue>	The first value that is recorded in the buffer index
<extraValue>	A second value that is recorded in the buffer index (only valid for buffer style FULLWRITable)
<seconds>	An integer that represents the seconds
<fractionalSeconds>	The portion of time that represents the fractional seconds
<status>	Information about the reading; see Details
<channel>	The channel to which to assign the data

Details

This command writes the data you specify into a reading buffer. The reading buffer must be set to the writable or full writable style, which is set when you make the buffer.

Data must be added in chronological order. If the time is not specified for a reading, it is set to one integer second after the last reading. As you write the data, the front-panel home screen updates and displays the reading you entered.

The <status> parameter provides additional information about the reading. The options are shown in the following table.

Buffer status bits for sense measurements		
Bit (hex)	Decimal	Description
0x0001	1	Measure status questionable
0x0006	6	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize)

0x0008	8	Measure terminal; front is 1, rear is 0
0x0010	16	Measure status limit 2 low
0x0020	32	Measure status limit 2 high
0x0040	64	Measure status limit 1 low
0x0080	128	Measure status limit 1 high
0x0100	256	First reading in a group
0x0200	512	Relative offset
0x0400	1024	Scan

Example 1

```
:TRAC:MAKE "write2me", 1000, WRITable
:TRAC:WRIT:FORM "write2me", WATT, 4
:TRAC:WRIT:READ "write2me", 1
:TRAC:WRIT:READ "write2me", 2
:TRAC:WRIT:READ "write2me", 3
:TRAC:WRIT:READ "write2me", 4
:TRAC:WRIT:READ "write2me", 5
:TRAC:WRIT:READ "write2me", 6
:TRAC:DATA? 1, 6, "write2me", read, unit
```

Creates a 1000-point reading buffer named `write2me`. Style is writable.

Set the data format to show a unit of watts with 4½ digit resolution.

Write 6 pieces of data into the buffer.

Read the buffer.

Output:

```
1.000000E+00,Watt DC,2.000000E+00,Watt DC,3.000000E+00,Watt DC,4.000000E+00,Watt
DC,5.000000E+00,Watt DC,6.000000E+00,Watt DC
```

Example 2

```
:TRAC:MAKE "write2me", 1000, FULLWRIT
:TRAC:WRIT:FORM "write2me", WATT, 4, WATT, 4
:TRAC:WRIT:READ "write2me", 1, 7
:TRAC:WRIT:READ "write2me", 2, 8
:TRAC:WRIT:READ "write2me", 3, 9
:TRAC:WRIT:READ "write2me", 4, 10
:TRAC:WRIT:READ "write2me", 5, 11
:TRAC:WRIT:READ "write2me", 6, 12
:TRAC:DATA? 1, 6, "write2me", read, unit, read, unit
```

Creates a 1000-point reading buffer named `write2me`. Style is full writable.

Set the data format to show units of watts with 4½ digit resolution for the first value and the second value in the buffer index.

Write 12 pieces of data into the buffer.

Read the buffer.

Output:

```
1.000000E+00,Watt DC,7.000000E+00,Watt DC,2.000000E+00,Watt DC,8.000000E+00,Watt
DC,3.000000E+00,Watt DC,9.000000E+00,Watt DC,4.000000E+00,Watt
DC,1.000000E+01,Watt DC,5.000000E+00,Watt DC,1.100000E+01,Watt
DC,6.000000E+00,Watt DC,1.200000E+01,Watt DC
```

Also see

[Reading buffers](#) (on page 6-1)

[:TRACe:DATA?](#) (on page 12-237)

[:TRACe:MAKE](#) (on page 12-243)

[:TRACe:WRITe:FORMat](#) (on page 12-264)

[Writable reading buffers](#) (on page 6-36)

TRIGger subsystem

The commands in this subsystem configure and control the trigger operations, including the trigger model.

:ABORt

This command stops all trigger model commands and scans on the instrument.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:ABORt

Details

When this command is received, the instrument stops the trigger model and scans.

Also see

[Aborting the trigger model](#) (on page 8-67)

[Trigger model](#) (on page 8-35)

:INITiate[:IMMediate]

This command starts the trigger model or scan.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:INITiate[:IMMediate]

Example

```
INIT
*WAI
```

Starts the trigger model or scan and then waits until the commands are complete to accept new commands.

Also see

[:ABORt](#) (on page 12-60)

[:TRIGger:PAUSe](#) (on page 12-333)

[:TRIGger:RESume](#) (on page 12-333)

[Trigger model](#) (on page 8-35)

:TRIGger:BLENder<n>:CLEar

This command clears the blender event detector and resets the overrun indicator of blender <n>.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:BLENder<n>:CLEar

<n>	The blender number: up to two

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector.

Example

:TRIG:BLEN2:CLE	Clears the event detector for blender 2.

Also see

None

:TRIGger:BLENder<n>:MODE

This command selects whether the blender performs OR operations or AND operations.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Trigger blender clear	Save settings	AND

Usage

```
:TRIGger:BLENder<n>:MODE <operation>
:TRIGger:BLENder<n>:MODE?
```

<n>	The blender number: up to two
<operation>	The type of operation: ■ OR ■ AND

Details

This command selects whether the blender waits for any one event (OR) or waits for all selected events (AND) before signaling an output event.

Example

```
:DIG:LINE3:MODE TRIG, IN
:DIG:LINE5:MODE TRIG, IN
:TRIG:BLEN1:MODE OR
:TRIG:BLEN1:STIM1 DIG3
:TRIG:BLEN1:STIM2 DIG5
```

Set digital I/O lines 3 and 5 as trigger in lines. Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

[:TRIGger:BLENder<n>:STIMulus<m>](#) (on page 12-272)

:TRIGger:BLENder<n>:OVERrun?

This command indicates whether or not an event was ignored because of the event detector state.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:BLENder<n>:OVERrun?
```

<n>	The blender number: up to two

Details

Indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector that is built into the event blender itself.

This command does not indicate if an overrun occurred in any other part of the trigger model or in any other trigger object that is monitoring the event. It also is not an indication of an action overrun.

Example

:TRIG:BLEN1:OVER?	If an event was ignored, the output is 1. If an event was not ignored, the output is 0.

Also see

[:TRIGger:BLENder<n>:CLEar](#) (on page 12-270)

:TRIGger:BLENder<n>:STIMulus<m>

This command specifies the events that trigger the blender.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle Trigger blender clear	Save settings	NONE

Usage

:TRIGger:BLENder<n>:STIMulus<m> <event>
:TRIGger:BLENder<n>:STIMulus<m>?

<n>	The blender number: up to two
<m>	The stimulus input number (1 to 4)
<event>	See Details

Details

There are four stimulus inputs that can each select a different event.

Use none to disable the blender input.

The <event> parameter may be any of the trigger events shown in the following table.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENder<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPlete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmlimit (returns NOT3)

Example

```
:DIG:LINE3:MODE TRIG, IN
:DIG:LINE5:MODE TRIG, IN
:TRIG:BLEN1:MODE OR
:TRIG:BLEN1:STIM1 DIG3
:TRIG:BLEN1:STIM2 DIG5
```

Set digital I/O lines 3 and 5 as trigger in lines. Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

[:TRIGger:BLENder<n>:MODE](#) (on page 12-271)

:TRIGger:BLOCK:BRANch:ALWays

This command defines a trigger model block that always goes to a specific block.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:BRANch:ALWays <blockNumber>, <branchToBlock>
```

<blockNumber>	The sequence of the block in the trigger model
<branchToBlock>	The block number of the trigger-model block to execute when the trigger model reaches this block

Details

When the trigger model reaches a branch-always building block, it goes to the building block set by <branchToBlock>.

Example

```
TRIG:BLOC:BRAN:ALW 9, 20
```

When the trigger model reaches block 9, it always branches to block 20.

Also see

None

:TRIGger:BLOCK:BRANch:COUNter

This command defines a trigger model block that branches to a specified block a specified number of times.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:BRANch:COUNter <blockNumber>, <targetCount>, <branchToBlock>
```

<blockNumber>	The sequence of the block in the trigger model
<targetCount>	The number of times to repeat
<branchToBlock>	The block number of the trigger model block to execute when the counter is less than to the <targetCount> value

Details

This command defines a trigger model building block that branches to another block using a counter to iterate a specified number of times.

Counters increment every time the trigger model reaches them until they are more than or equal to the count value. At that point, the trigger model continues to the next building block in the sequence.

If you are using remote commands, you can query the counter. The counter is incremented immediately before the branch compares the actual counter value to the set counter value. Therefore, the counter is at 0 until the first comparison. When the trigger model reaches the set counter value, branching stops and the counter value is one greater than the setting. Use `:TRIGger:BLOCk:BRANch:COUNter:COUNt?` to query the counter.

Example

```
TRIG:LOAD "EMPTY"
TRIG:BLOC:BUFF:CLEAR 1
TRIG:BLOC:MDIG 2
TRIG:BLOC:BRAN:COUN 3, 5, 2
TRIG:BLOC:DEL:CONS 4, 1
TRIG:BLOC:BRAN:COUN 5, 3, 2
```

Reset trigger model settings.
Clear `defbuffer1` at the beginning of the trigger model.
Loop and make five readings.
Delay a second.
Loop three more times back to block 2.
At end of execution, 15 readings are stored in `defbuffer1`.

Also see

[:TRIGger:BLOCk:BRANch:COUNter:COUNt?](#) (on page 12-275)

:TRIGger:BLOCk:BRANch:COUNter:COUNt?

This command returns the count value of the trigger model counter block.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

`:TRIGger:BLOCk:BRANch:COUNter:COUNt? <blockNumber>`

<blockNumber>	The sequence of the block in the trigger model

Details

This command returns the counter value. When the counter is active, this returns the present count. If the trigger model has started or is running but has not yet reached the counter block, this value is 0.

Example

```
*RST
TRIG:BLOC:BUFF:CLEAR 1
TRIG:BLOC:MDIG 2
TRIG:BLOC:DEL:CONS 3, 0.1
TRIG:BLOC:BRAN:COUN 4, 10, 2
INIT

TRIG:BLOCK:BRAN:COUN:COUN? 4
*WAI
```

Reset trigger model settings.
Clear defbuffer1 at the beginning of the trigger model.
Loop and make five readings.
Delay 0.1 s.
Loop ten more times back to block 2.
Send the count command to check the count that has been completed for block 4.
At end of execution, 10 readings are stored in defbuffer1.

Also see

[:TRIGger:BLOCK:BRANch:COUNter](#) (on page 12-274)

:TRIGger:BLOCK:BRANch:COUNter:RESet

This command creates a block in the trigger model that resets a branch counter to 0.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

:TRIGger:BLOCK:BRANch:COUNter:RESet <blockNumber>, <counter>

<blockNumber>	The sequence of the block in the trigger model
<counter>	The block number of the counter that is to be reset

Details

When the trigger model reaches the Counter Reset block, it resets the count of the specified Branch on Counter block to zero.

Example

```
TRIG:LOAD "EMPTY"
TRIG:BLOC:BUFF:CLEAR 1
TRIG:BLOC:MDIG 2
TRIG:BLOC:BRAN:COUN 3, 5, 2
TRIG:BLOC:DEL:CONS 4, 1
TRIG:BLOC:BRAN:COUN 5, 3, 2
TRIG:BLOC:BRAN:COUN:RES 6, 3
```

Reset trigger model settings.
Clear defbuffer1 at the beginning of the trigger model.
Loop and make five readings.
Delay a second.
Loop three more times back to block 2.
Reset block 3 to 0.

Also see

[:TRIGger:BLOCK:BRANch:COUNter](#) (on page 12-274)

[:TRIGger:BLOCK:BRANch:COUNter:COUNt?](#) (on page 12-275)

:TRIGger:BLOCK:BRANch:DELTA

This command defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:BRANch:DELTA <blockNumber>, <targetDifference>, <branchToBlock>
:TRIGger:BLOCK:BRANch:DELTA <blockNumber>, <targetDifference>, <branchToBlock>,
    <measureDigitizeBlock>
```

<blockNumber>	The sequence of the block in the trigger model
<targetDifference>	The value against which the block compares the difference between the measurements
<branchToBlock>	The block number of the trigger model block to execute when the difference between the measurements is less than or equal to the <targetDifference>
<measureDigitizeBlock>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

This block calculates the difference between the last two measurements from a measure/digitize block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measure/digitize block, it will compare measurements of a measure/digitize block that precedes the branch delta block. For example, if you have a measure/digitize block, a wait block, another measure/digitize block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure/digitize block. If a preceding measure/digitize block does not exist, an error occurs.

Example

TRIG:BLOC:BRAN:DELT 5, 0.5, 7, 4	Configure trigger block 5 to compare the differences between the measurements made in block 4. If the difference between them is less the 0.5, branch to block 7.

Also see

[Delta block](#) (on page 8-52)

:TRIGger:BLOCK:BRANch:EVENT

This command branches to a specified block when a specified trigger event occurs.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

:TRIGger:BLOCK:BRANch:EVENT <blockNumber>, <event>, <branchToBlock>

<blockNumber>	The sequence of the block in the trigger model
<event>	The event that must occur before the trigger model branches the specified block
<branchToBlock>	The block number of the trigger model block to execute when the specified event occurs

Details

The branch-on-event block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

If you set the branch event to none, an error is generated when you run the trigger model.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a Notify block before the On Event block. Set the Notify block to use the same timer as the On Event block.

The following table shows the constants for the events.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
Analog trigger	ATRigger
Trigger event blender <n> (up to two), which combines trigger events	BLENDER<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMMANd
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Front-panel TRIGGER key press	DISPlay
External in trigger	EXTernal
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
No trigger event	NONE
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>

Example

```
:TRIG:BLOC:BRAN:EVEN 6, DISP, 2
```

When the trigger model reaches this block, if the front-panel TRIGGER key has been pressed, the trigger model returns to block 2. If the TRIGGER key has not been pressed, the trigger model continues to block 7 (the next block in the trigger model).

Also see

[On event block](#) (on page 8-52)

:TRIGger:BLOCK:BRANch:LIMit:CONStant

This command defines a trigger model block that goes to a specified block if a measurement meets preset criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:BRANch:LIMit:CONStant <blockNumber>, <limitType>, <limitA>,
<limitB>, <branchToBlock>
:TRIGger:BLOCK:BRANch:LIMit:CONStant <blockNumber>, <limitType>, <limitA>,
<limitB>, <branchToBlock>, <measureDigitizeBlock>
```

<blockNumber>	The sequence of the block in the trigger model
<limitType>	The type of limit (ABOVe, BELow, INside, or OUTside)
<limitA>	The limit that the measurement is tested against; if <i>limitType</i> is set to: - ABOVe: This value is ignored - BELow: The measurement must be below this value - INside: The low limit that the measurement is compared against - OUTside: The low limit that the measurement is compared against
<limitB>	The upper limit that the measurement is tested against; if <i>limitType</i> is set to: - ABOVe: The measurement must be above this value - BELow: This value is ignored - INside: The high limit that the measurement is compared against - OUTside: The high limit that the measurement is compared against
<branchToBlock>	The block number of the trigger model block to execute when the measurement meets the defined criteria
<measureDigitizeBlock>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-constant-limits block goes to a branching block if a measurement meets the criteria set by this command.

The type of limit can be:

- Above: The measurement is above the value set by limit B; limit A must be set, but is ignored when this type is selected
- Below: The measurement is below the value set by limit A; limit B must be set, but is ignored when this type is selected
- Inside: The measurement is inside the values set by limits A and B; limit A must be the low value and Limit B must be the high value
- Outside: The measurement is outside the values set by limits A and B; limit A must be the low value and Limit B must be the high value

The measurement block must be a measure/digitize block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from a measure/digitize block is used.

If the limit A is more than the limit B, the values are automatically swapped so that the lesser value is used as the lower limit.

Example

```
TRIGger:BLOCK:BRANch:LIMit:CONStant 5, OUTside, 0.15, 0.65, 8
```

Configure trigger block 5 to check for measurements in the last measure/digitize block. If the measurements are outside of the 0.15 and 0.65 limits, branch to block 8.

Also see

[Constant Limit block](#) (on page 8-48)

:TRIGger:BLOCK:BRANch:LIMit:DYNamic

This command defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:BRANch:LIMit:DYNamic <blockNumber>, <limitType>, <limitNumber>,  
    <branchToBlock>  
:TRIGger:BLOCK:BRANch:LIMit:DYNamic <blockNumber>, <limitType>, <limitNumber>,  
    <branchToBlock>, <measureDigitizeBlock>
```

<blockNumber>	The sequence of the block in the trigger model
<limitType>	The type of limit (ABOVe, BELow, INside, or OUTside)
<limitNumber>	The limit number (1 or 2)
<branchToBlock>	The block number of the trigger model block to execute when the limits are met
<measureDigitizeBlock>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-dynamic-limits block defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

When you define this block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- The limit number (you can have 1 or 2 limits)
- The block to go to if the measurement meets the criteria
- The block that makes the measurement that is compared to the limits; the last measurement from that block is used

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values, which are set using the front-panel Calculations limit settings or through commands. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

Limit values are stored in the measure configuration list, so you can use a configuration list to step through different limit values.

The measure/digitize block must occur in the trigger model before the branch-on-dynamic-limits block. If no block is defined, the measurement from the previous measure/digitize block is used. If no previous measure/digitize block exists, an error is reported.

Example

```
CALC2:LIM1:STAT ON
CALC2:LIM1:LOW -5.17
CALC2:LIM1:UPP -4.23
TRIG:BLOC:BRAN:LIM:DYN 9, IN, 1, 12, 7
```

Set the limits on with a low limit of -5.17 and a high limit of -4.23. Set trigger block 9 to test if the limit is inside those limits based on the measurement reading at block 7. If the measurement is within the limits, go to block 12.

Also see

[Dynamic Limit block](#) (on page 8-50)

[:CALCulate2:<function>:LIMit<Y>:LOWer\[:DATA\]](#) (on page 12-24)

[:CALCulate2:<function>:LIMit<Y>:UPPer\[:DATA\]](#) (on page 12-27)

:TRIGger:BLOCK:BRANch:ONCE

This command causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

:TRIGger:BLOCK:BRANch:ONCE <blockNumber>, <branchToBlock>

<blockNumber>	The sequence of the block in the trigger model
<branchToBlock>	The block number of the trigger model block to execute when the trigger model first encounters this block

Details

The branch-once building block branches to a specified block the first time trigger model execution encounters the branch-once block. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

The once block is reset when trigger model execution reaches the idle state. Therefore, the branch-once block always executes the first time the trigger model execution encounters this block.

Example

```
:TRIG:BLOC:BRAN:ONCE 2, 4
```

The first time the trigger model reaches block 2, the trigger model goes to block 4 instead of proceeding to the default sequence of block 3.

Also see

[Once block](#) (on page 8-51)
[:TRIGger:BLOCK:BRANch:ONCE:EXCLuded](#) (on page 12-286)

:TRIGger:BLOCK:BRANch:ONCE:EXCLuded

This command causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

:TRIGger:BLOCK:BRANch:ONCE:EXCLuded <blockNumber>, <branchToBlock>

<blockNumber>	The sequence of the block in the trigger model
<branchToBlock>	The block number of the trigger model block to execute when the trigger model encounters this block after the first encounter

Details

The branch-once-excluded block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts or is placed in idle.

Example

:TRIG:BLOC:BRAN:ONCE:EXCL 2, 4

When the trigger model reaches block 2 the first time, the trigger model goes to block 3. If the trigger model reaches this block again, the trigger model goes to block 4.

Also see

- [Once excluded block](#) (on page 8-51)
- [:TRIGger:BLOCK:BRANch:ONCE](#) (on page 12-285)

:TRIGger:BLOCK:BUFFEr:CLEar

This command defines a trigger model block that clears the reading buffer.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:BUFFEr:CLEar <blockNumber>  
:TRIGger:BLOCK:BUFFEr:CLEar <blockNumber>, "<bufferName>"
```

<blockNumber>	The sequence of the block in the trigger model
<bufferName>	The name of the buffer, which must be an existing buffer; if no buffer is defined, defbuffer1 is used

Details

When trigger model execution reaches the buffer clear trigger block, the instrument empties the specified reading buffer. The specified buffer can be the default buffer or a buffer that you defined.

Example

```
TRIG:LOAD "EMPTY"  
TRIG:BLOC:BUFF:CLE 1  
TRIG:BLOC:MDIG 2  
TRIG:BLOC:BRAN:COUN 3, 5, 2  
TRIG:BLOC:DEL:CONS 4, 1  
TRIG:BLOC:BRAN:COUN 5, 3, 2
```

Reset trigger model settings.
Clear defbuffer1 at the beginning of the trigger model.
Loop and make 5 readings.
Delay 1 s.
Loop three more times back to block 2.
At end of execution, 15 readings are stored in defbuffer1.

Also see

- [Buffer clear block](#) (on page 8-42)
- [:TRACe:MAKE](#) (on page 12-243)

:TRIGger:BLOCK:CONFig:NEXT

This command recalls the settings at the next index of a configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

:TRIGger:BLOCK:CONFig:NEXT <blockNumber>, "<configurationList>"

<blockNumber>	The sequence of the block in the trigger model
<configurationList>	A string that defines the configuration list to recall

Details

When trigger model execution reaches a configuration recall next block, the settings at the next index in the specified configuration list are restored.

The first time the trigger model encounters this block for a specific configuration list, the first index is recalled. Each subsequent time this block is encountered, the settings at the next index in the configuration list are recalled and take effect before the next step executes. When the last index in the list is reached, it returns to the first index.

The configuration list must be defined before you can use this block.

Example

TRIG:BLOC:CONF:NEXT 12, "SETTINGS_LIST"	Set trigger block 12 to restore the settings from the next index that is stored in the configuration list SETTINGS_LIST.

Also see

[Configuration lists](#) (on page 4-99)

:TRIGger:BLOCK:CONFig:PREVious

This command defines a trigger model block that recalls the settings stored at the previous index in a configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:CONFig:PREVious <blockNumber>, "<configurationList>"
```

<blockNumber>	The sequence of the block in the trigger model
<configurationList>	A string that defines the configuration list to recall

Details

The Config List Prev block defines a trigger model block that recalls the settings stored at the previous index in a configuration list.

The configuration list previous index trigger block type recalls the previous index in a configuration list. It configures the settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

The first time the trigger model encounters this block, the last index in the configuration list is recalled. Each subsequent time trigger model execution reaches a configuration list previous block for this configuration list, it goes backward one index. When the first index in the list is reached, it goes to the last index in the configuration list.

The configuration list must be defined before you can use this block.

Example

TRIG:BLOC:CONF:PREV 14, "SETTINGS_LIST"	Set trigger block 14 to restore the settings from the previous index that is stored in the configuration list SETTINGS_LIST.

Also see

[Configuration lists](#) (on page 4-99)

:TRIGger:BLOCK:CONFig:RECall

This command recalls the system settings that are stored in a configuration list.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:CONFig:RECall <blockNumber>, "<configurationList>"  
:TRIGger:BLOCK:CONFig:RECall <blockNumber>, "<configurationList>", <index>
```

<blockNumber>	The sequence of the block in the trigger model
<configurationList>	A string that defines the configuration list to recall
<index>	The index in the configuration list to recall; default is 1

Details

When the trigger model reaches a configuration recall block, the settings in the specified configuration list are recalled.

You can restore a specific set of configuration settings in the configuration list by defining the index.

The configuration list must be defined before you can use this block. If the configuration list changes, verify that the trigger model count is still accurate.

Example

TRIG:BLOCK:CONF:RECALL 1, "SETTINGS_LIST", 1	Recall the settings in index 1 of the SETTINGS_LIST configuration list as block 1 of the trigger model.

Also see

[Configuration lists](#) (on page 4-99)
[\[:SENSe\[1\]\]:CONFiguration:LIST:STORE](#) (on page 12-183)

:TRIGger:BLOCK:DElay:CONStant

This command adds a constant delay to the execution of a trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

:TRIGger:BLOCK:DElay:CONStant <blockNumber>, <time>

<blockNumber>	The sequence of the block in the trigger model
<time>	The amount of time to delay (167 ns to 10 ks; 0 for no delay)

Details

When trigger model execution reaches a delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made, and if any previously executed block started infinite measurements, they also continue to be made.

This delay waits for the delay time to elapse before proceeding to the next block in the trigger model.

If other delays have been set, this delay is in addition to the other delays.

Example

TRIG:LOAD "EMPTY" TRIG:BLOC:BUFF:CLEAR 1 TRIG:BLOC:MDIG 2 TRIG:BLOC:BRAN:COUN 3, 5, 2 TRIG:BLOC:DEL:CONS 4, 1 TRIG:BLOC:BRAN:COUN 5, 3, 2	Reset trigger model settings. Clear defbuffer1 at the beginning of the trigger model. Loop and make 5 readings. Delay a second. Loop three more times back to block 2. At end of execution, 15 readings are stored in defbuffer1.

Also see

None

:TRIGger:BLOCk:DELaY:DYNamic

This command adds a user delay to the execution of the trigger model.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

:TRIGger:BLOCk:DELaY:DYNamic <blockNumber>, MEASure<n>

<blockNumber>	The sequence of the block in the trigger model
<n>	The number of the user delay; 1 to 5 set by [:SENSe[1]]:<function>:DELaY:USER<n>

Details

When trigger model execution reaches a dynamic delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made.

Each measure function can have up to five unique user delay times (M1 to M5). Digitize user delays are handled as measure user delays, so you can have a total of five measure and digitize user delays. The delay time is set by the user-delay command, which is only available over a remote interface.

Example

FUNC "VOLT" :VOLT:DEL:USER1 5 :TRIG:LOAD "EMPTY" :TRIG:BLOC:BUFF:CLEAR 1 :TRIG:BLOC:MDIG 2 :TRIG:BLOC:BRAN:COUN 3, 5, 2 :TRIG:BLOC:DEL:DYN 4, MEAS1 :TRIG:BLOC:BRAN:COUN 5, 3, 2 :INIT	Set function to DC voltage. Set user delay 1 for DC voltage measurements to 5 s. Clear the trigger model. Set trigger block 1 to clear the reading buffer. Set trigger block 2 to make or digitize a measurement. Set trigger block 3 to loop and make or digitize five measurements. Set trigger block 4 to a dynamic delay, using user delay 1. Set trigger block 5 to loop three times. Start the trigger model.

Also see

[\[:SENSe\[1\]\]:<function>:DELaY:USER<n>](#) (on page 12-139)

:TRIGger:BLOCK:DIGital:IO

This command defines a trigger model block that sets the lines on the digital I/O port high or low.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:DIGital:IO <blockNumber>, <bitPattern>  
:TRIGger:BLOCK:DIGital:IO <blockNumber>, <bitPattern>, <bitMask>
```

<blockNumber>	The sequence of the block in the trigger model
<bitPattern>	Sets the value that specifies the output line bit pattern: 0 to 63
<bitMask>	Specifies the bit mask; if omitted, all lines are driven: 0 to 63

Details

To set the lines on the digital I/O port high or low, you can send a bit pattern that is specified as an integer value. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The bit mask defines the bits in the pattern that are driven high or low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63) or omit this parameter. If the bit for a line in the bit pattern is set to 1, the line is driven high. If the bit is set to 0 in the bit pattern, the line is driven low.

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

:DIGital:LINE3:MODE DIG,OUT :DIGital:LINE4:MODE DIG,OUT :DIGital:LINE5:MODE DIG,OUT :DIGital:LINE6:MODE DIG,OUT :TRIG:BLOC:DIG:IO 4, 20, 60	The first four lines of code configures digital I/O lines 3 through 6 as digital outputs. Trigger block 4 is then configured with a bit pattern of 20 (digital I/O lines 3 and 5 high). The optional bit mask is specified as 60 (lines 3 through 6), so both lines 3 and 5 are driven high.

Also see

[:DIGital:LINE<n>:MODE](#) (on page 12-44)
[Digital I/O bit weighting](#) (on page 8-13)
[Digital I/O port configuration](#) (on page 8-3)

:TRIGger:BLOCK:LIST?

This command returns the settings for all trigger model blocks.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:BLOCK:LIST?

Details

This returns the settings for the trigger model.
If a scan is set up, this returns two trigger models that begin with START and END blocks.

Example

:TRIG:BLOC:LIST?

Returns the settings for the trigger model. Example output is:
1) BUFFER_CLEAR BUFFER: defbuffer1
2) DELAY_CONSTANT DELAY: 0.001000000
3) MEASURE_DIGITIZE BUFFER: defbuffer1 INITIAL MODE: MEAS INITIAL COUNT: 1
4) BRANCH_COUNTER VALUE: 11 BRANCH_BLOCK: 2

Also see

None

:TRIGger:BLOCK:LOG:EVENT

This command allows you to log an event in the event log when the trigger model is running.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

:TRIGger:BLOCK:LOG:EVENT <blockNumber>, <eventNumber>, "<message>"

<blockNumber>	The sequence of the block in the trigger model
<eventNumber>	The event number: ■ INFO<n> ■ WARNIng<n> ■ ERRor<n> Where <n> is 1 to 4; you can define up to four of each type You can also set ABORT, which aborts the trigger model immediately and posts a warning event log message
<message>	A string up to 31 characters

Details

This block allows you to log an event in the event log when trigger model execution reaches this block. You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

You can define the type of event (information, warning, abort model, or error). All events generated by this block are logged in the event log. Warning and error events are also displayed in a popup on the front-panel display.

Note that using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger model blocks.

Example

```
TRIGger:BLOCK:LOG:EVENT 9, INFO2, "Trigger model complete"
Set trigger-model block 9 to log an event when the trigger model completes.
```

Also see

None

:TRIGger:BLOCK:MDIGitize

This command defines a trigger block that makes or digitizes a measurement.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:MDIGitize <blockNumber>
:TRIGger:BLOCK:MDIGitize <blockNumber>, "<bufferName>"
:TRIGger:BLOCK:MDIGitize <blockNumber>, "<bufferName>", <count>
```

<blockNumber>	The sequence of the block in the trigger model
<bufferName>	The name of the buffer, which must be an existing buffer; if no buffer is defined, defbuffer1 is used
<count>	Specifies the number of readings to make before moving to the next block in the trigger model; set to: - A specific value (default is 1 if nothing set) - Infinite (run continuously until stopped): INF - Stop infinite to stop the count: 0 - Use most recent count value (default): AUTO

Details

This block triggers measurements based on the measure function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

1. The instrument begins triggering measurements.
2. The trigger model execution waits for the measurement to be made.
3. The instrument processes the reading and places it into the specified reading buffer.

If you are defining a user-defined reading buffer, you must create it before you define this block.

When you set the count to a finite value, trigger model execution does not proceed until all operations are complete.

If you set the count to infinite, the trigger model executes subsequent blocks when the measurement is made; the triggering of measurements continues in the background until the trigger model execution reaches another measure/digitize block or until the trigger model ends. To use infinite, there must be a block after the measure/digitize block in the trigger model, such as a wait block. If there is no subsequent block, the trigger model stops, which stops measurements.

When you set the count to auto, the trigger model uses the count value that is active for the selected function instead of a specific value. You can use this with configuration lists to change the count value each time a measure/digitize block is encountered.

NOTE

If you bring in code that uses a measure or digitize block and does not define the count, the count is set to 1. For example, `:TRIGger:BLOCk:MEASure 1, "defbuffer1"` changes to `:TRIGger:BLOCk:MDIGitize 1, "defbuffer1", 1`.

Example 1

```
TRIG:LOAD "EMPTY"
TRIG:BLOC:BUFF:CLEAR 1, "defbuffer2"
TRIG:BLOC:MDIG 2, "defbuffer2"
TRIG:BLOC:BRAN:COUN 3, 5, 2
TRIG:BLOC:DEL:CONS 4, 1
TRIG:BLOC:BRAN:COUN 5, 3, 2
INIT
*WAI
TRAC:ACT? "defbuffer2"
```

Reset trigger model settings.
Clear `defbuffer2` at the beginning of the trigger model.
Set the measurements to be stored in `defbuffer2`.
Loop and make five readings.
Delay 1 s.
Loop three more times back to block 2.
At end of execution, 15 readings are stored in `defbuffer2`.
Output:
15

Example 2

```
*RST
SENS:CONF:LIST:CRE "countactive"
COUN 2
SENSe:CONF:LIST:STOR "countactive"
COUN 10
SENSe:CONF:LIST:STOR "countactive"
COUN 3
SENSe:CONF:LIST:STOR "countactive"

TRIG:BLOC:CONF:NEXT 1, "countactive"
TRIG:BLOC:MDIG 2, "defbuffer1", AUTO
TRIG:BLOC:DEL:CONS 3, 1
TRIG:BLOC:BRAN:COUN 4, 3, 1

INIT
*WAI
TRAC:ACT? "defbuffer1"
```

Reset the instrument.

Set up a configuration list named `countactive`.

Set the measure count to 2. (If you are digitizing, replace `COUN` with `DIG:COUN`.)

Store the count in index 1.

Set the measure count to 10.

Store the count in index 2.

Set the measure count to 3.

Store the count in index 3.

Set up trigger-model block 1 to call the next index from the `countactive` configuration list.

Set block 2 to measure or digitize and store the readings in `defbuffer1` and to use the active count.

Set block 3 to add a delay of 1 s.

Set block 4 to iterate through the trigger model three times, returning to block 1.

Start the trigger model.

Output the number of readings. There should be 15 readings.

Also see

[Measure/Digitize block](#) (on page 8-40)

[:TRACe:MAKE](#) (on page 12-243)

:TRIGger:BLOCK:NOP

This command creates a placeholder that performs no action in the trigger model; available only using remote commands.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:NOP <blockNumber>
```

<blockNumber>	The sequence of the block in the trigger model

Details

If you remove a trigger model block, you can use this block as a placeholder for the block number so that you do not need to renumber the other blocks.

Example

TRIG:BLOC:NOP 5	Set block number 5 to be a no operation block.

Also see

None

:TRIGger:BLOCK:NOTify

This command defines a trigger model block that generates a trigger event and immediately continues to the next block.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:NOTify <blockNumber>, <notifyID>
```

<blockNumber>	The sequence of the block in the trigger model
<notifyID>	The identification number of the notification: 1 to 8

Details

When trigger model execution reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. This assigns a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

NOTE

The TSP-Link and digital I/O options require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

When you call this event, you use the format `NOTIFY` followed by the notify identification number. For example, if you assign `<notifyID>` as 4, you would refer to it as `NOTIFY4` in the command that references it.

Example

```
:TRIG:BLOC:NOT 5, 2
:TRIG:BLOC:BRAN:EVEN 6, NOTIFY2, 2
```

Define trigger-model block 5 to be the notify 2 event. Assign the notify 2 event to be the trigger for stimulus for the branch event for block 6.

Also see

Notify block

:TRIGger:BLOCK:WAIT

This command defines a trigger model block that waits for an event before allowing the trigger model to continue.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:BLOCK:WAIT <blockNumber>, <event>
:TRIGger:BLOCK:WAIT <blockNumber>, <event>, <clear>
:TRIGger:BLOCK:WAIT <blockNumber>, <event>, <clear>, <logic>, <event>
:TRIGger:BLOCK:WAIT <blockNumber>, <event>, <clear>, <logic>, <event>, <event>
```

<blockNumber>	The sequence of the block in the trigger model
<event>	The event that must occur before the trigger block allows trigger execution to continue; see Details for event names
<clear>	To clear previously detected trigger events when entering the wait block: ENTER To immediately act on any previously detected triggers and not clear them (default): NEVER
<logic>	If each event must occur before the trigger model continues: AND If at least one of the events must occur before the trigger model continues: OR

Details

You can use the wait block to synchronize measurements with other instruments and devices.

The event can occur before trigger model execution reaches the wait block. If the event occurs after trigger model execution starts but before the trigger model execution reaches the wait block, the trigger model records the event. By default, when trigger model execution reaches the wait block, it executes the wait block without waiting for the event to happen again (the clear parameter is set to never).

The instrument clears the memory of the recorded event when trigger model execution is at the start block and when the trigger model exits the wait block. It also clears the recorded trigger event when the clear parameter is set to enter.

All items in the list are subject to the same action; you cannot combine **AND** and **OR** logic in a single block.

You cannot leave the first event as no trigger. If the first event is not defined, the trigger model errors when you attempt to initiate it.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a notify block before the wait block. Set the notify block to use the same timer as the wait block.

The following table shows the constants for the events.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
Analog trigger	ATRIgger
Trigger event blender <n> (up to two), which combines trigger events	BLENDer<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGIo<n>
Front-panel TRIGGER key press	DISPlay
External in trigger	EXTernal
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
No trigger event	NONE
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>

Example 1

```
:TRIGger:BLOCK:WAIT 9, DISP
```

Set trigger-model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing and to act on a recorded TRIGGER key event that gets detected either before or after reaching block 9.

Example 2

:TRIGger:BLOCK:WAIT 9, DISP, ENTer	Set trigger-model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing and to act only on a recorded TRIGGER key event that gets detected when block 9 is reached.

Also see

[Wait block](#) (on page 8-38)

:TRIGger:CONTInuous

This command determines the trigger mode setting after bootup.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Nonvolatile memory Save settings	AUTO

Usage

TRIGger:CONTInuous <setting>
TRIGger:CONTInuous?

<setting>	Do not start continuous measurements after bootup: OFF Start continuous measurements after bootup: AUTO Place the instrument into local control and start continuous measurements after bootup: REStart

Details

Conditions must be valid before continuous measurements can start.

When the restart parameter is selected, the instrument is placed in local mode, aborts any running scripts, and aborts any trigger models that are running. If the command is in a script, it is the last command that runs before the script is aborted. The restart parameter is not stored in nonvolatile memory, so it does not affect start up behavior.

The off and automatic parameters are stored in nonvolatile memory, so they affect start up behavior.

Example

TRIG:CONT OFF	When the instrument starts up, the Measurement Method is set to idle.

Also see

None

:TRIGger:DIGital<n>:IN:CLEar

This command clears the trigger event on a digital input line.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:DIGital<n>:IN:CLEar
```

<n>	Digital I/O trigger line: 1 to 6

Details

The event detector of a trigger enters the detected state when an event is detected. For the specified trigger line, this command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to 0).

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

:TRIG:DIG2:IN:CLE	Clears the trigger event detector on I/O line 2.

Also see

[:DIGital:LINE<n>:MODE](#) (on page 12-44)

[Digital I/O port configuration](#) (on page 8-3)

[:TRIGger:DIGital<n>:IN:OVERrun?](#) (on page 12-304)

:TRIGger:DIGital<n>:IN:EDGE

This command sets the edge used by the trigger event detector on the given trigger line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	FALL

Usage

```
:TRIGger:DIGital<n>:IN:EDGE <detectedEdge>
```

```
:TRIGger:DIGital<n>:IN:EDGE?
```

<n>	Digital I/O trigger line: 1 to 6
<detectedEdge>	The trigger edge value: ■ Detect falling-edge triggers as inputs: FALLing ■ Detect rising-edge triggers as inputs: RISing ■ Detect either falling or rising-edge triggers as inputs: EITHer See Details for descriptions of values

Details

This command sets the logic on which the trigger event detector and the output trigger generator operate on the specified trigger line.

To directly control the line state, set the mode of the line to digital and use the write command. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse.

Trigger mode values

Value	Description
FALLing	Detects falling-edge triggers as input when the line is configured as an input or open drain.
RISing	Detects rising-edge triggers as input when the line is configured as an open drain.
EITHer	Detects rising- or falling-edge triggers as input when the line is configured as an input or open drain.

Example

:DIG:LINE4:MODE TRIG, IN :TRIG:DIG4:IN:EDGE RIS	Sets the input trigger mode for the digital I/O line 4 to detect rising-edge triggers as input.

Also see

[Digital I/O port configuration](#) (on page 8-3)
[:DIGital:LINE<n>:MODE](#) (on page 12-44)
[:DIGital:WRITe <n>](#) (on page 12-48)
[:TRIGger:DIGital<n>:IN:CLEar](#) (on page 12-303)

:TRIGger:DIGital<n>:IN:OVERrun?

This command returns the event detector overrun status.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:DIGital<n>:IN:OVERrun?

<n>	Digital I/O trigger line: 1 to 6

Details

This command returns the event detector overrun status as 0 (false) or 1 (true).

If this is 1, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

TRIG:DIG1:IN:OVER?	Returns 0 if no overruns have occurred or 1 if one or more overruns have occurred for I/O line 1.

Also see

[Digital I/O port configuration](#) (on page 8-3)
[:DIGital:LINE<n>:MODE](#) (on page 12-44)

:TRIGger:DIGital<n>:OUT:LOGic

This command sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NEG

Usage

:TRIGger:DIGital<n>:OUT:LOGic <logicType>
:TRIGger:DIGital<n>:OUT:LOGic?

<n>	Digital I/O trigger line: 1 to 6
<logicType>	The output logic of the trigger generator: - Assert a TTL-high pulse for output: POSitive - Assert a TTL-low pulse for output: NEGative

Details

This command sets the trigger event generator to assert a TTL pulse for output logic. Positive is a high pulse; negative is a low pulse.

Example

:DIG:LINE4:MODE TRIG, OUT :TRIG:DIG4:OUT:LOG NEG	Sets line 4 mode to be a trigger output and sets the output logic of the trigger event generator to negative (asserts a low pulse).

Also see

[:DIGital:LINE<n>:MODE](#) (on page 12-44)
[Digital I/O port configuration](#) (on page 8-3)

:TRIGger:DIGital<n>:OUT:PULSewidth

This command describes the length of time that the trigger line is asserted for output triggers.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	10e-6 (10 μ s)

Usage

```
:TRIGger:DIGital<n>:OUT:PULSewidth <width>  
:TRIGger:DIGital<n>:OUT:PULSewidth?
```

<n>	Digital I/O trigger line: 1 to 6
<width>	Pulse length (0 to 100 ks)

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely.

Example

DIG:LINE1:MODE TRIG, OUT TRIG:DIG1:OUT:PULS 2	Set digital line 1 to trigger out. Set the pulse to 2 s.

Also see

[:DIGital:LINE<n>:MODE](#) (on page 12-44)
[:DIGital:WRITe <n>](#) (on page 12-48)
[Digital I/O port configuration](#) (on page 8-3)

:TRIGger:DIGital<n>:OUT:STIMulus

This command selects the event that causes a trigger to be asserted on the digital output line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:DIGital<n>:OUT:STIMulus <event>  
:TRIGger:DIGital<n>:OUT:STIMulus?
```

<n>	Digital I/O trigger line: 1 to 6
<event>	The event to use as a stimulus; see Details

Details

The digital trigger pulsewidth command determines how long the trigger is asserted.

The trigger stimulus for a digital I/O line can be set to one of the trigger events that are described in the following table.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENDER<n>
Trigger timer <n> (1 to 4) expired	TIMER<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPLete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmlimit (returns NOT3)

Example 1

```
:TRIG:DIG2:OUT:STIMulus TIM3
```

Set the stimulus for output digital trigger line 2 to be the expiration of trigger timer 3.

Example 2

```
*RST
SENS:FUNC 'RES', (@101:109)
ROUT:SCAN:CRE (@101:109)
ROUT:SCAN:COUN:SCAN 10
ROUT:SCAN:BYPASS ON

ROUT:SCAN:CHAN:STIM DIG1
DIG:LINE1:MODE TRIG, IN
TRIG:DIG1:IN:EDGE FALL
DIG:LINE3:MODE TRIG, OUT
TRIG:DIG3:OUT:LOG NEG
TRIG:DIG3:OUT:STIM SCANCHAN
INIT
```

Reset the instrument.
Set channels 101 through 109 to measure 2-wire resistance.
Create a scan using channels 101 through 109.
Set the scan count to 10.
Bypass the first channel close trigger.
Set the channel close stimulus to respond to a falling edge trigger coming in on digital input line 1.
Set a digital output signal to trigger a negative pulse each time a defined scan channel is closed.
Initiate the scan.

Also see

[Digital I/O port configuration](#) (on page 8-3)
[:DIGital:LINE<n>:STATe](#) (on page 12-46)
[:TRIGger:DIGital<n>:OUT:LOGic](#) (on page 12-305)

:TRIGger:EXTErnal:IN:CLEar

This command clears the trigger event on the EXTERNAL TRIGGER IN line.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:EXTErnal:IN:CLEar
```

Details

The event detector of a trigger enters the detected state when an event is detected. This command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to false).

Example

```
:TRIG:EXT:IN:CLE
```

Clears the trigger event detector on the EXTERNAL TRIGGER IN line.

Also see

[:TRIGger:EXTErnal:IN:OVERrun?](#) (on page 12-310)

:TRIGger:EXTernal:IN:EDGE

This command sets the type of edge that is detected as an input on the EXTERNAL TRIGGER IN trigger line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	FALL

Usage

```
:TRIGger:EXTernal:IN:EDGE <detectedEdge>
```

```
:TRIGger:EXTernal:IN:EDGE?
```

<detectedEdge>

The trigger edge value:

- Detect falling-edge triggers as inputs: FALLing
- Detect rising-edge triggers as inputs: RISing
- Detect either falling or rising-edge triggers as inputs: EITHer

See **Details** for descriptions of values

Details

The input state of EXTERNAL TRIGGER IN is controlled by the type of edge specified by this command.

Trigger mode values

Value	Description
FALLing	Detects falling-edge triggers as input
RISing	Detects rising-edge triggers as input
EITHer	Detects rising- or falling-edge triggers as input

Example

```
:TRIG:EXT:IN:EDGE RIS
```

Sets the EXTERNAL TRIGGER IN line to detect rising-edge triggers as inputs.

Also see

[:TRIGger:EXTernal:OUT:LOGic](#) (on page 12-311)

[:TRIGger:EXTernal:OUT:STIMulus](#) (on page 12-312)

:TRIGger:EXTernal:IN:OVERrun?

This command returns the event detector overrun status.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:EXTernal:IN:OVERrun?

Details

This command returns the event detector overrun status as 0 (false) or 1 (true).

If this is 1, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

TRIG:EXT:IN:OVER?	Returns 0 if no overruns have occurred or 1 if one or more overruns have occurred for the EXTERNAL TRIGGER IN line.

Also see

None

:TRIGger:EXTernal:OUT:LOGic

This command sets the output logic of the trigger event generator to positive or negative for the EXTERNAL TRIGGER OUT line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NEG

Usage

```
:TRIGger:EXTernal:OUT:LOGic <logicType>  
:TRIGger:EXTernal:OUT:LOGic?
```

<logicType>	The output logic of the trigger generator: - Assert a TTL-high pulse for output: POSitive - Assert a TTL-low pulse for output: NEGative

Details

This command sets the trigger event generator to assert a TTL pulse for output logic. Positive is a high pulse; negative is a low pulse.

Example

<pre>*RST :TRIG:EXT:IN:CLE :TRIG:EXT:OUT:LOG NEG :TRIG:EXT:OUT:STIM EXT :TRIG:EXT:IN:EDGE FALL</pre>	Reset the EXTERNAL TRIGGER IN and OUT line values to their defaults. Clear any event triggers on the EXTERNAL TRIGGER IN line. Set the output logic to negative (it asserts a low pulse). Set the stimulus to the EXTERNAL TRIGGER IN line. Set the external input to detect a falling edge.

Also see

None

:TRIGger:EXTernal:OUT:STIMulus

This command selects the event that causes a trigger to be asserted on the EXTERNAL TRIGGER OUT line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:EXTernal:OUT:STIMulus <event>  
:TRIGger:EXTernal:OUT:STIMulus?
```

<event>	The event to use as a stimulus; see Details

Details

The trigger stimulus for the EXTERNAL TRIGGER OUT line can be set to one of the trigger events described in the following table.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command <code>device_trigger</code>	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENder<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPlete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmLimit (returns NOT3)

Example 1

:TRIG:EXT:OUT:STIM TIM3

Set the stimulus for the EXTERNAL TRIGGER OUT line to be the expiration of trigger timer 3.

Example 2

*RST
SENS:FUNC 'RES', (@101:109)
ROUT:SCAN:CRE (@101:109)
ROUT:SCAN:COUN:SCAN 10
ROUT:SCAN:MEAS:STIM EXT
TRIG:EXT:IN:EDGE FALL
TRIG:EXT:OUT:LOG NEG
TRIG:EXT:OUT:STIM SCANMEAS
INIT

Reset the instrument.
Set channels 101 through 109 to measure 2-wire resistance.
Create a scan using channels 101 through 109.
Set the scan count to 10.
Set the channel measurement stimulus to be triggered by a falling edge pulse on the EXTERNAL TRIGGER IN line.
Set the EXTERNAL TRIGGER OUT line to generate a negative pulse each time a scan channel makes a measurement.
Initiate the scan.

Also see

[:TRIGger:EXTeRnal:OUT:LOGic](#) (on page 12-311)

:TRIGger:LAN<n>:IN:CLEar

This command clears the event detector for a LAN trigger.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:LAN<n>:IN:CLEar

<n>

The LAN event number: 1 to 8

Details

The trigger event detector enters the detected state when an event is detected. This function clears a trigger event detector and discards the history of the trigger packet.

This function clears all overruns associated with this LAN trigger.

Example

:TRIG:LAN5:IN:CLE

Clears the event detector with LAN packet 5.

Also see

[:TRIGger:LAN<n>:IN:OVERrun?](#) (on page 12-316)

:TRIGger:LAN<n>:IN:EDGE

This command sets the trigger operation and detection mode of the specified LAN event.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	EITH

Usage

```
:TRIGger:LAN<n>:IN:EDGE <mode>  
:TRIGger:LAN<n>:IN:EDGE?
```

<n>	The LAN event number: 1 to 8
<mode>	The trigger mode; see the Details for more information

Details

This command controls how the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values		
Mode	Trigger packets detected as input	LAN trigger packet generated for output with a...
EITHer	Rising or falling edge (positive or negative state)	negative state
FALLing	Falling edge (negative state)	negative state
RISing	Rising edge (positive state)	positive state

Example

:TRIG:LAN2:IN:EDGE FALL	Set the LAN trigger mode for event 2 to falling edge.

Also see

- [Digital I/O](#) (on page 8-1)
- [TSP-Link System Expansion Interface](#) (on page 9-1)

:TRIGger:LAN<n>:IN:OVERrun?

This command indicates the overrun status of the LAN event detector.

Type	Affected by	Where saved	Default value
Query only	LAN trigger clear	Not applicable	Not applicable

Usage

:TRIGger:LAN<n>:IN:OVERrun?

<n>	The LAN event number: 1 to 8

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event.

It also is not an indication of an output trigger overrun.

The trigger overrun state for the specified LAN packet is returned as 1 (true) or 0 (false).

Example

TRIG:LAN5:IN:OVER?	Checks the overrun status of a trigger on LAN5 and outputs the value, such as: 0

Also see

[:TRIGger:LAN<n>:IN:CLEAr](#) (on page 12-314)

:TRIGger:LAN<n>:OUT:CONNeCT:STATe

This command prepares the event generator for outgoing trigger events.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:LAN<n>:OUT:CONNeCT:STATe <state>  
:TRIGger:LAN<n>:OUT:CONNeCT:STATe?
```

<n>	The LAN event number: 1 to 8
<state>	Do not send event messages: OFF or 0 Prepare to send event messages: ON or 1

Details

When this is set to **ON**, the instrument prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the protocol or IP address for this event is changed.

When this is set to **OFF**, for TCP connections, this closes the TCP connection.

Example

```
:TRIGger:LAN1:OUT:PROTOcol MULT  
:TRIGger:LAN1:OUT:CONNeCT:STATe ON
```

Set the protocol to multicast and prepare the event generator to send event messages.

Also see

[:TRIGger:LAN<n>:OUT:IP:ADDReSS](#) (on page 12-317)

[:TRIGger:LAN<n>:OUT:PROTOcol](#) (on page 12-319)

:TRIGger:LAN<n>:OUT:IP:ADDReSS

This command specifies the address (in dotted-decimal format) of UDP or TCP listeners.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	"0.0.0.0"

Usage

```
:TRIGger:LAN<n>:OUT:IP:ADDReSS "<address>"  
:TRIGger:LAN<n>:OUT:IP:ADDReSS?
```

<n>	The LAN event number: 1 to 8
<address>	A string that represents the LAN address in dotted decimal notation

Details

Sets the IP address for outgoing trigger events.

After you change this setting, you must send the connect command before outgoing messages can be sent.

Example

TRIG:LAN1:OUT:IP:ADDR "192.0.32.10"	Use IP address 192.0.32.10 to connect the LAN trigger.

Also see

[:TRIGger:LAN<n>:OUT:CONNeCT:STATe](#) (on page 12-317)

:TRIGger:LAN<n>:OUT:LOGic

This command sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NEG

Usage

:TRIGger:LAN<n>:OUT:LOGic <logicType>
:TRIGger:LAN<n>:OUT:LOGic?

<n>	The LAN event number: 1 to 8
<logicType>	The type of logic: ■ POSitive ■ NEGative

Example

TRIG:LAN1:OUT:LOG POS	Set the logic to positive.

Also see

None

:TRIGger:LAN<n>:OUT:PROTOcol

This command sets the LAN protocol to use for sending trigger messages.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	TCP

Usage

```
:TRIGger:LAN<n>:OUT:PROTOcol <protocol>  
:TRIGger:LAN<n>:OUT:PROTOcol?
```

<n>	The LAN event number: 1 to 8
<protocol>	The protocol to use for messages from the trigger: ■ TCP ■ UDP ■ MULTicast

Details

The LAN trigger listens for trigger messages on all the supported protocols. However, it uses the designated protocol for sending outgoing messages.

After you change this setting, you must re-connect the LAN trigger event generator before you can send outgoing event messages.

When multicast is selected, the trigger IP address is ignored, and event messages are sent to the multicast address 224.0.23.159.

Example

:TRIG:LAN1:OUT:PROT TCP :TRIG:LAN1:OUT:CONN:STAT	Set the LAN protocol for trigger messages to be TCP and re-connect the LAN trigger event generator.

Also see

- [:TRIGger:LAN<n>:OUT:CONNeCT:STATe](#) (on page 12-317)
- [:TRIGger:LAN<n>:OUT:IP:ADDReSS](#) (on page 12-317)

:TRIGger:LAN<n>:OUT:STIMulus

This command specifies events that cause this trigger to assert.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:LAN<n>:OUT:STIMulus <LANevent>  
:TRIGger:LAN<n>:OUT:STIMulus?
```

<n>	A number specifying the trigger packet over the LAN for which to set or query the trigger source: 1 to 8
<LANevent>	The LAN event that causes this trigger to assert

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set the event to one of the existing trigger events, which are shown in the following table.

Setting this attribute to none disables automatic trigger generation.

If any events are detected before the trigger LAN connection is sent, the event is ignored, and the action overrun is set.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENder<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPlete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmLimit (returns NOT3)

Example

```
TRIG:LAN1:OUT:STIM TIM1
```

Set the timer 1 trigger event as the source for the LAN packet 1 trigger stimulus.

Also see

[:TRIGger:LAN<n>:OUT:CONNeCT:STATe](#) (on page 12-317)

:TRIGger:LOAD "ConfigList"

This command loads a trigger-model template configuration that uses measure configuration lists.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "ConfigList", "<measureConfigList>"  
:TRIGger:LOAD "ConfigList", "<measureConfigList>", <delay>  
:TRIGger:LOAD "ConfigList", "<measureConfigList>", <delay>, "<bufferName>"
```

<measureConfigList>	A string that contains the name of the measurement configuration list to use
<delay>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

This trigger-model template incorporates a configuration list. You must set up the configuration lists before loading the trigger model. If the configuration lists change, you must resend this command.

You can also set a delay and change the reading buffer.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the TRIGger:BLOCK:LIST? command to view the trigger-model blocks in a list format.

Example

```
*RST  
:SENS:CONF:LIST:CRE "MEASURE_LIST"  
:SENS:CURRE:RANG 1e-3  
:SENSe:CONF:LIST:STOR "MEASURE_LIST"  
:SENS:CURRE:RANG 10e-3  
:SENSe:CONF:LIST:STOR "MEASURE_LIST"  
:SENS:CURRE:RANG 100e-3  
:SENSe:CONF:LIST:STOR "MEASURE_LIST"  
:TRIG:LOAD "ConfigList", "MEASURE_LIST"  
INIT
```

Set up a configuration list named MEASURE_LIST.
Load the configuration list trigger model, using this configuration list.
Start the trigger model.

Also see

[:TRIGger:BLOCK:LIST?](#) (on page 12-294)

:TRIGger:LOAD "DurationLoop"

This command loads a trigger-model template configuration that makes continuous measurements for a specified amount of time.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "DurationLoop", <duration>  
:TRIGger:LOAD "DurationLoop", <duration>, <delay>  
:TRIGger:LOAD "DurationLoop", <duration>, <delay>, "<readingBuffer>"
```

<duration>	The time to make measurements: 500 ns to 100 ks
<delay>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<readingBuffer>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

When you load this trigger-model template, you can specify amount of time to make a measurement and the length of the delay before the measurement.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the TRIGger:BLOCK:LIST? command to view the trigger-model blocks in a list format.

Example

*RST SENS:FUNC "CURR" TRIG:LOAD "DurationLoop", 10, 0.01 INIT	Reset the instrument. Set the instrument to measure DC current. Load the DurationLoop trigger model to make measurements for 10 s with a 10 ms delay before each measurement. Start the trigger model.

Also see

[:TRIGger:BLOCK:LIST?](#) (on page 12-294)

:TRIGger:LOAD "Empty"

This command resets the trigger model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:LOAD "Empty"

Details

When you load this trigger-model template, any blocks that have been defined in the trigger model are cleared so the trigger model has no blocks defined.

Example

```
TRIG:LOAD "Empty"  
TRIG:BLOC:BUFF:CLEAR 1  
TRIG:BLOC:MDIG 2  
TRIG:BLOC:BRAN:COUN 3, 5, 2  
TRIG:BLOC:DEL:CONS 4, 1  
TRIG:BLOC:BRAN:COUN 5, 3, 2  
TRAC:ACT? "defbuffer1"
```

Reset trigger model settings.
Clear defbuffer1 at the beginning of execution of the trigger model.
Loop and take 5 readings.
Delay 1 s.
Loop three more times back to block 2.
At the end of execution, 15 readings are stored in defbuffer1.
Output:
15

Also see

None

:TRIGger:LOAD "GradeBinning"

This command loads a trigger-model template configuration that sets up a grading operation.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>, <limit2Pattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>, <limit2Pattern>, <limit3High>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>,
    <limit3Pattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>,
    <limit3Pattern>, <limit4High>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>,
    <limit3Pattern>, <limit4High>, <limit4Low>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>,
    <limit3Pattern>, <limit4High>, <limit4Low>, <limit4Pattern>
:TRIGger:LOAD "GradeBinning", <components>, <startInLine>, <startDelay>,
    <endDelay>, <limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>,
    <limit2High>, <limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>,
    <limit3Pattern>, <limit4High>, <limit4Low>, <limit4Pattern>, "<bufferName>"
```

<components>	The number of components to measure (1 to 268,435,455)
<startInLine>	The input line that starts the test; 5 for digital line 5, 6 for digital line 6, or 7 for external in; default is 5
<startDelay>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks

<endDelay>	The delay time after the measurement; default is 0 for no delay: 167 ns to 10 ks
<limitxHigh>	x is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
<limitxLow>	x is 1, 2, 3, or 4; the lower limit that the measurement is compared against
<limit1Pattern>	The bit pattern that is sent when the measurement fails limit 1; range 1 to 15; default is 1
<limit2Pattern>	The bit pattern that is sent when the measurement fails limit 2; range 1 to 15; default is 2
<limit3Pattern>	The bit pattern that is sent when the measurement fails limit 3; range 1 to 15; default is 4
<limit4Pattern>	The bit pattern that is sent when the measurement fails limit 4; range 1 to 15; default is 8
<allPattern>	The bit pattern that is sent when all limits have passed; 1 to 15; default is 15
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

This trigger-model template allows you to grade components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the pass pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `TRIGger:BLOCK:LIST?` command to view the trigger-model blocks in a list format.

Also see

[:TRIGger:BLOCK:LIST?](#) (on page 12-294)

:TRIGger:LOAD "LogicTrigger"

This command loads a trigger-model template configuration that sets up an external or digital trigger through the digital I/O.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "LogicTrigger", <digInLine>, <digOutLine>, <count>
:TRIGger:LOAD "LogicTrigger", <digInLine>, <digOutLine>, <count>, <clear>
:TRIGger:LOAD "LogicTrigger", <digInLine>, <digOutLine>, <count>, <clear>, <delay>
:TRIGger:LOAD "LogicTrigger", <digInLine>, <digOutLine>, <count>, <clear>, <delay>,
  "<bufferName>"
```

<digInLine>	The digital input line (1 to 6) or external input line (7); also, the event that the trigger model will wait on in block 1
<digOutLine>	The digital output line (1 to 6) or external input line (7)
<count>	The number of measurements the instrument will make
<clear>	To clear previously detected trigger events when entering the wait block: ENTer To immediately act on any previously detected triggers and not clear them (default): NEVer
<delay>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<bufferName>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; default is defbuffer1

Details

This trigger model waits for a digital input or external trigger input event to occur, makes a measurement, and issues a notify event. If a digital output line is selected, a notify event asserts a digital output line. A notify event asserts the external trigger output line regardless of the line settings. You can set the line to 7 to assert only the external trigger output line, or to another setting to assert both a digital output line and the external trigger output line.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the TRIGger:BLOCK:LIST? command to view the trigger-model blocks in a list format.

Example

```
:TRIGger:LOAD "LogicTrigger", 7, 2, 10, 0.001, "defbuffer1"
```

Set up the template to use the external trigger in line and wait for a pulse to trigger measurements. Pulse digital out line 2 when the measurement is complete. The external trigger output line is also pulsed. Make 10 measurements, with a delay of 1 ms before each measurement. Store the measurements in defbuffer1.

Also see

[:TRIGger:BLOCK:LIST?](#) (on page 12-294)

[:TRIGger:DIGital<n>:OUT:LOGic](#) (on page 12-305)

:TRIGger:LOAD "LoopUntilEvent"

This command loads a trigger-model template configuration that makes continuous measurements until the specified event occurs.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <delay>
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <delay>,
    "<bufferName>"
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <clear>
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <clear>, <delay>
:TRIGger:LOAD "LoopUntilEvent", <eventConstant>, <position>, <clear>, <delay>,
    "<bufferName>"
```

<eventConstant>	The event that ends infinite triggering or readings set to occur before the trigger; see Details
<position>	The number of readings to make in relation to the size of the reading buffer; enter as a percentage: 0 to 100
<clear>	To clear previously detected trigger events when entering the wait block (default): ENTer To immediately act on any previously detected triggers and not clear them: NEVer
<delay>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

The event constant is the event that ends infinite triggering or ends readings set to occur before the trigger and start post-trigger readings. The trigger model makes readings until it detects the event constant. After the event, it makes a finite number of readings, based on the setting of the trigger position.

The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled. For example, if this is set to 75 for a reading buffer that holds 10,000 readings, the trigger model makes 2500 readings after it detects the source event. There are 7500 pre-trigger readings and 2500 post-trigger readings.

The instrument makes two sets of readings. The first set is made until the trigger event occurs. The second set is made after the trigger event occurs, up to the number of readings calculated by the position parameter.

You cannot have the event constant set at none when you run this trigger-model template.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

You can use the TRIGger:BLOCK:LIST? command to view the trigger-model blocks in a list format.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 8); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMMand
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENDer<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
Analog trigger	ATRigger
External in trigger	EXTernal

Example

<pre>*RST SENS:FUNC "CURR" TRIG:LOAD "LoopUntilEvent", DISP, 25 INIT</pre>	Reset the instrument. Set the instrument to measure DC current. Set the LoopUntilEvent trigger model to make measurements until the front-panel TRIGGER key is pressed after starting the trigger model, then make measurements that constitute 75% of the reading buffer. Start the trigger model.

Also see

[:TRIGger:BLOCK:LIST?](#) (on page 12-294)

:TRIGger:LOAD "SimpleLoop"

This command loads a trigger-model template configuration.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "SimpleLoop", <count>  
:TRIGger:LOAD "SimpleLoop", <count>, <delay>  
:TRIGger:LOAD "SimpleLoop", <count>, <delay>, "<bufferName>"
```

<count>	The number of measurements the instrument will make
<delay>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

This command sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you define in the Count parameter.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen.

You can use the TRIGger:BLOCK:LIST? command to view the trigger-model blocks in a list format.

Example

*RST SENS:FUNC "CURR" SENS:CURR:RANG:AUTO ON TRIG:LOAD "SimpleLoop", 10 INIT *WAI TRAC:DATA? 1, 10, "defbuffer1", READ, REL	Reset the instrument and set it to measure current with automatic range setting. Set a simple trigger loop with a count of 10. Start the trigger model. Postpone execution of subsequent commands until all previous commands are finished. Read data and return the reading and relative time.

Also see

[:TRIGger:BLOCK:LIST?](#) (on page 12-294)

:TRIGger:LOAD "SortBinning"

This command loads a trigger-model template configuration that sets up a sorting operation.

Type	Affected by	Where saved	Default value
Command only	Recall settings Instrument reset Power cycle	Save settings	Not applicable

Usage

```
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>, <limit2Pattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>, <limit2Pattern>, <limit3High>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
<limit4High>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
<limit4High>, <limit4Low>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
<limit4High>, <limit4Low>, <limit4Pattern>
:TRIGger:LOAD "SortBinning", <components>, <startInLine>, <startDelay>, <endDelay>,
<limit1High>, <limit1Low>, <limit1Pattern>, <allPattern>, <limit2High>,
<limit2Low>, <limit2Pattern>, <limit3High>, <limit3Low>, <limit3Pattern>,
<limit4High>, <limit4Low>, <limit4Pattern>, "<bufferName>"
```

<components>	The number of components to measure
<startInLine>	The input line that starts the test; 5 for digital line 5, 6 for digital line 6, or 7 for external in; default is 5
<startDelay>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<endDelay>	The delay time after the measurement; default is 0 for no delay: 167 ns to 10 ks
<limitxHigh>	x is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against

<limitxLow>	x is 1, 2, 3, or 4; the lower limit that the measurement is compared against
<limit1Pattern>	The bit pattern that is sent when the measurement passes limit 1; range 1 to 15; default is 1
<limit2Pattern>	The bit pattern that is sent when the measurement passes limit 2; range 1 to 15; default is 2
<limit3Pattern>	The bit pattern that is sent when the measurement passes limit 3; range 1 to 15; default is 4
<limit4Pattern>	The bit pattern that is sent when the measurement passes limit 4; range 1 to 15; default is 8
<allPattern>	The bit pattern that is sent when all limits have failed: 1 to 15; default is 15
<bufferName>	A string that indicates the reading buffer; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

This trigger-model template allows you to sort components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the all fail pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `TRIGger:BLOCK:LIST?` command to view the trigger-model blocks in a list format.

Also see

[:TRIGger:BLOCK:LIST?](#) (on page 12-294)

:TRIGger:PAUSE

This command pauses a running scan or trigger model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:PAUSE

Details

This command pauses the scan or trigger model.

To continue the trigger model and the scan, send the resume command.

Example

INIT TRIG:PAUS TRIG:RES *WAI	Start a trigger model or scan, then pause and resume it.

Also see

[:INITiate:IMMediate](#) (on page 12-60)

[:TRIGger:RESume](#) (on page 12-333)

:TRIGger:RESume

This command continues a paused scan or trigger model.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:RESume

Details

This command continues running the scan or trigger model operation if the scan or trigger model was paused.

Example

INIT TRIG:PAUS TRIG:RES *WAI	Start a trigger model or scan, then pause and resume it.

Also see

[:INITiate:IMMediate](#) (on page 12-60)

[:TRIGger:PAUSE](#) (on page 12-333)

:TRIGger:STATe?

This command returns the present state of the trigger model.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

:TRIGger:STATe?

Details

This command returns the state of the trigger model. The instrument checks the state of a started trigger model every 100 ms.

This command returns the trigger state and the block that the trigger model last executed. If the trigger model supports a scan, three states and two block numbers are returned.

The trigger model states are:

- **Idle:** The trigger model is stopped
- **Running:** The trigger model is running
- **Waiting:** The trigger model has been in the same wait block for more than 100 ms
- **Empty:** The trigger model is selected, but no blocks are defined
- **Paused:** The trigger model is paused
- **Building:** Blocks have been added
- **Failed:** The trigger model is stopped because of an error
- **Aborting:** The trigger model is stopping
- **Aborted:** The trigger model is stopped

Example

:TRIG:STAT?

An example output if the trigger model is inactive and ended at block 9 is:
IDLE; IDLE; 9

Also see

None

:TRIGger:TIMer<n>:CLEar

This command clears the timer event detector and overrun indicator for the specified trigger timer number.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:TIMer<n>:CLEar
```

<n>	Trigger timer number: 1 to 4

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

:TRIG:TIM1:CLE	Clears trigger timer 1.

Also see

[:TRIGger:TIMer<n>:COUNT](#) (on page 12-335)

[:TRIGger:TIMer<n>:STARt:OVERrun?](#) (on page 12-339)

:TRIGger:TIMer<n>:COUNT

This command sets the number of events to generate each time the timer generates a trigger event or is enabled as a timer or alarm.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1

Usage

```
:TRIGger:TIMer<n>:COUNT <count>
```

```
:TRIGger:TIMer<n>:COUNT?
```

<n>	Trigger timer number: 1 to 4
<count>	The number of times to repeat the trigger (0 to 1,048,575)

Details

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

Example 1

```
TRIG:TIM2:COUN 4
```

Set the number of events to generate for trigger timer 2 to four.

Example 2

```
*RST
TRIG:TIM4:DEL 0.5
TRIG:TIM4:STAR:STIM NOT8
TRIG:TIM4:STAR:GEN OFF
TRIG:TIM4:COUN 20
TRIG:TIM4:STAT ON

TRIG:LOAD "Empty"
TRIG:BLOC:BUFF:CLEAR 1, "defbuffer1"
TRIG:BLOC:NOT 2, 8
TRIG:BLOC:WAIT 3, TIM4
TRIG:BLOC:MDIG 4
TRIG:BLOC:BRAN:COUN 5, 20, 3
INIT
*WAI
TRAC:ACT? "defbuffer1"
```

Set trigger timer 4 to have a 0.5 s delay.
 Set the stimulus for trigger timer 4 to be the notify 8 event.
 Set the trigger timer 4 stimulus to off.
 Set the timer event to occur when the timer delay elapses.
 Set the trigger timer 4 count to 20.
 Enable trigger timer 4.

Clear the trigger model.
 Set trigger-model block 1 to clear the buffer.
 Set trigger-model block 2 to generate the notify 8 event.
 Set trigger-model block 3 to wait for trigger timer 4 to occur.
 Set trigger-model block 4 to make a reading and store it in default buffer 1.
 Set trigger-model block 5 to repeat the trigger model 20 times, starting at block 3.
 Start the trigger model.
 Output the number of entries in default buffer 1.

Output:
 20

Also see

[:TRIGger:TIMer<n>:CLEar](#) (on page 12-335)

[:TRIGger:TIMer<n>:DELay](#) (on page 12-337)

:TRIGger:TIMer<n>:DELay

This command sets and reads the timer delay.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	1e-5 (10 μ s)

Usage

```
:TRIGger:TIMer<n>:DELay <interval>  
:TRIGger:TIMer<n>:DELay?
```

<n>	Trigger timer number: 1 to 4
<interval>	Delay interval (8 μ s to 100 ks)

Details

A delay is the period after the timer is triggered and before the timer generates a trigger event. Each time the timer is triggered, it uses this delay period.

If you use the trigger timer with a trigger model, make sure the trigger timer delay is set so that the readings are paced correctly.

Reading this command returns the delay interval that will be used the next time the timer is triggered.

Example

```
TRIG:TIM2:DEL 50E-6
```

 Set trigger timer 2 to delay for 50 μ s.

Also see

None

:TRIGger:TIMer<n>:STARt:FRActional

This command configures an alarm or a time in the future when the timer will start.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0

Usage

```
:TRIGger:TIMer<n>:STARt:FRActional <time>  
:TRIGger:TIMer<n>:STARt:FRActional?
```

<n>	Trigger timer number: 1 to 4
<time>	The time in fractional seconds: 0 to <1 s

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time in the past or if it is in the future.

Example

<pre>TRIG:TIM1:STAR:SEC 60 TRIG:TIM1:START:FRAC 0.5 TRIG:TIM1:STAT ON</pre>	Set the timer for 60.5 s. Enable the trigger timer for timer 1.

Also see

[:TRIGger:TIMer<n>:STARt:SEConds](#) (on page 12-339)
[:TRIGger:TIMer<n>:STATe](#) (on page 12-342)

:TRIGger:TIMer<n>:STARt:GENerate

This command specifies when timer events are generated.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (OFF)

Usage

```
:TRIGger:TIMer<n>:STARt:GENerate <state>
:TRIGger:TIMer<n>:STARt:GENerate?
```

<n>	Trigger timer number: 1 to 4
<state>	Generate a timer event when the timer delay elapses: OFF or 0 Generate a timer event when the timer starts and when the delay elapses: ON or 1

Details

When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses. This generates the event TIMERN.

Example

<pre>TRIG:TIM3:STAR:GEN ON</pre>	Set trigger timer 3 to generate an event when the timer starts and when the timer delay elapses.

Also see

None

:TRIGger:TIMer<n>:STARt:OVERrun?

This command indicates if an event was ignored because of the event detector state.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
:TRIGger:TIMer<n>:STARt:OVERrun?
```

<n>	Trigger timer number: 1 to 4

Details

This command indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

This returns 0 if there is no overrun or 1 if there is an overrun.

Example

TRIG:TIM1:STAR:OVER?	Checks the overrun status on trigger timer 1.

Also see

None

:TRIGger:TIMer<n>:STARt:SEConds

This command configures an alarm or a time in the future when the timer will start.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	0 (0 s)

Usage

```
:TRIGger:TIMer<n>:STARt:SEConds <time>
```

```
:TRIGger:TIMer<n>:STARt:SEConds?
```

<n>	Trigger timer number: 1 to 4
<time>	The time: 0 to 2,147,483,647 s

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time that has passed.

Example

```
TRIG:TIM1:STAR:SEC 60
TRIG:TIM1:STAR:FRAC 0.5
TRIG:TIM1:STAT ON
```

Set the timer for 60.5 s.
Enable the trigger timer for timer 1.

Also see

[:TRIGger:TMer<n>:STATe](#) (on page 12-342)

:TRIGger:TMer<n>:STARt:STIMulus

This command describes the event that starts the trigger timer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	NONE

Usage

```
:TRIGger:TMer<n>:STARt:STIMulus <event>
:TRIGger:TMer<n>:STARt:STIMulus?
```

<n>	Trigger timer number: 1 to 4
<event>	The event that starts the trigger timer; see Details

Details

Set the stimulus to any trigger event to start the timer when that event occurs.

Set the stimulus to none to disable event processing and use the timer as a timer or alarm based on the start time.

Trigger events are described in the table below.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event	NONE
Front-panel TRIGGER key press	DISPlay
Notify trigger block <n> (1 to 3); the trigger model generates a trigger event when it executes the notify block	NOTify<n>
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	COMManD
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <n> (1 to 6)	DIGio<n>
Line edge detected on TSP-Link synchronization line <n> (1 to 3)	TSPLink<n>
Appropriate LXI trigger packet is received on LAN trigger object <n> (1 to 8)	LAN<n>
Trigger event blender <n> (up to two), which combines trigger events	BLENDer<n>
Trigger timer <n> (1 to 4) expired	TIMer<n>
External in trigger	EXTernal
Channel closed	SCANCHANnel (returns NOT6)
Scan completed	SCANCOMPLete (returns NOT8)
Measure completed	SCANMEASure (returns NOT7)
Notify trigger block generates a trigger event if a value in the scan is out of limits	SCANALARmlimit (returns NOT3)

Example

```
*RST
DIG:LINE1:MODE TRIG,IN
DIG:LINE2:MODE TRIG,OUT
TRIG:TIM1:DEL 35e-3
TRIG:TIM1:STAR:STIM DIG1
TRIG:DIG2:OUT:STIM TIM1
```

Reset the instrument to default settings. Set digital I/O line 1 for use as a trigger input.
Set digital I/O line 2 for use as a trigger output.
Set timer 1 to delay 35 ms.
Set timer 1 to start delaying once the digital I/O 1 event is detected.
Set digital I/O line 2 to output a pulse once the timer 1 event is detected.

Also see

None

:TRIGger:TIMer<n>:STATe

This command enables the trigger timer.

Type	Affected by	Where saved	Default value
Command and query	Recall settings Instrument reset Power cycle	Save settings	OFF (0)

Usage

```
:TRIGger:TIMer<n>:STATe <state>
:TRIGger:TIMer<n>:STATe?
```

<n>	Trigger timer number: 1 to 4
<state>	Disable the trigger timer: OFF or 0 Enable the trigger timer: ON or 1

Details

When this command is set to on, the timer performs the delay operation.

When this command is set to off, there is no timer on the delay operation.

You must enable a timer before it can use the delay settings or the alarm configuration. For expected results from the timer, it is best to disable the timer before changing a timer setting, such as delay or start seconds.

To use the timer as a simple delay or pulse generator with digital I/O lines, make sure the timer start time in seconds and fractional seconds is configured for a time in the past. To use the timer as an alarm, configure the timer start time in seconds and fractional seconds for the desired alarm time.

NOTE

The following examples require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Example 1

```
DIG:LINE3:MODE TRIG,OUT
TRIG:DIG3:OUT:STIM TIM2
SYSTem:TIME?
TRIG:TIM2:START:SECONDS <current time> + 60
TRIG:TIM2:STAT ON
```

To configure timer 2 for an alarm to fire 1 minute from now and output a pulse on digital I/O line 3, query to get the current time. Add 60 s to that value and use that to configure the start seconds. Enable the timer.

Example 2

```
*RST
DIG:LINE5:MODE TRIG,OUT
TRIG:DIG5:OUT:STIM TIM3
TRIG:TIM3:DEL 3e-3
TRIG:TIM3:COUNt 5
TRIG:TIM3:STAT ON
```

Configure timer 3 to generate five pulses on digital I/O line 5 that are 3 ms apart.

Example 3

```
*RST
DIG:LINE3:MODE TRIG,IN
DIG:LINE5:MODE TRIG,OUT
TRIG:DIG5:OUT:STIM TIM3
TRIG:TIM3:DEL 3e-3
TRIG:TIM3:COUNT 5
TRIG:TIM3:START:STIM DIG3
TRIG:TIM3:STAT ON
```

Configure timer 3 to generate 5 pulses on digital I/O line 5 that are 3 ms apart when a digital input is detected on digital line 3.

Also see

None

Introduction to TSP commands

In this section:

Introduction to TSP operation.....	13-1
Fundamentals of scripting for TSP	13-4
Fundamentals of programming for TSP	13-16
Test Script Builder.....	13-38
Memory considerations for the runtime environment	13-50

Introduction to TSP operation

Instruments that are enabled for Test Script Processor (TSP®) operate like conventional instruments by responding to a sequence of commands sent by the controller. You can send individual commands to the TSP-enabled instrument the same way you do when using any other instrument.

Unlike conventional instruments, TSP-enabled instruments can execute automated test sequences independently, without an external controller. You can load a series of TSP commands into the instrument using a remote computer or the front-panel port with a USB flash drive. You can store these commands as a script that can be run later by sending a single command message to the instrument.

You do not have to choose between using conventional control or script control. You can combine these forms of instrument control in the way that works best for your test application.

Controlling the instrument by sending individual command messages

The simplest method of controlling an instrument through the communications interface is to send it a message that contains remote commands. You can use a test program that resides on a computer (the controller) to sequence the actions of the instrument.

TSP commands can be function-based or attribute-based. Function-based commands are commands that control actions or activities. Attribute-based commands define characteristics of an instrument feature or operation.

Constants and enumerated types are commands that represent fixed values.

Functions

Function-based commands control actions or activities. A function-based command performs an immediate action on the instrument.

Each function consists of a function name followed by a set of parentheses (). Only include information in the parentheses if the function takes a parameter. If the function takes one or more parameters, they are placed between the parentheses and separated by commas.

Example 1

```
beeper.beep(0.5, 2400)
delay(0.250)
beeper.beep(0.5, 2400)
```

Emit a double beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Example 2

You can use the results of a function-based command directly or assign the results to variables for later access. The following code defines `x` and prints it.

```
x = math.abs(-100)
print(x)
```

Output:
100

Attributes

Attribute-based commands are commands that set the characteristics of an instrument feature or operation. For example, a characteristic of TSP-enabled instruments is the model number (`localnode.model`).

Attributes can be read-only, read-write, or write-only. They can be used as a parameter of a function or assigned to another variable.

To set the characteristics, attribute-based commands define a value. For many attributes, the value is in the form of a number, enumerated type, or a predefined constant.

Example 1: Set an attribute using a number

```
testData = buffer.make(500)
testData.capacity = 600
```

Use a function to create a buffer named `testData` with a capacity of 500, then use the `bufferVar.capacity` attribute to change the capacity to 600.

Example 2: Set an attribute using an enumerated type

```
display.lightstate = display.STATE_LCD_75
```

This attribute controls the brightness of the front-panel display and buttons. Setting this attribute to `display.STATE_LCD_75` sets the brightness of the display and buttons to 75% of full brightness.

Example 3: Set an attribute using a constant

```
format.data = format.REAL64
```

Using the constant `REAL64` sets the print format to double precision floating point format.

Reading an attribute

To read an attribute, you can use the attribute as the parameter of a function or assign it to another variable.

Example 1: Read an attribute using a function

```
print(display.lightstate)
```

Reads the status of the light state by passing the attribute to the print function. If the display light state is set to 50%, the output is: `display.STATE_LCD_50`

Example 2: Read an attribute using a variable

```
light = display.lightstate
print(light)
```

This reads the light state by assigning the attribute to a variable named `light`. If the display light state is set to 25%, the output is: `display.STATE_LCD_25`

Queries

Test Script Processor (TSP®) enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` and `printnumber()` commands generate output in the form of response messages. Each `print()` command creates one response message.

Example

```
x = 10  
print(x)
```

Example of an output response message:

10

Note that your output may be different if you set your ASCII precision setting to a different value.

USB flash drive path

You can use the file commands to open and close directories and files, write data, or to read a file on an installed USB flash drive.

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

`"/usb1/"`

Information on scripting and programming

If you need information about using scripts with your TSP-enabled instrument, see [Fundamentals of scripting for TSP](#) (on page 13-4).

If you need information about using the Lua programming language with the instrument, see [Fundamentals of programming for TSP](#) (on page 13-16).

Fundamentals of scripting for TSP

NOTE

Though it can improve your process to use scripts, you do not have to create scripts to use the instrument. Most of the examples in the documentation can be run by sending individual command messages. The next few sections of the documentation describe scripting and programming features of the instrument. You only need to review this information if you are using scripting and programming.

Scripting helps you combine commands into a block of code that the instrument can run. Scripts help you communicate with the instrument more efficiently.

Scripts offer several advantages compared to sending individual commands from the host controller (computer):

- Scripts are easier to save, refine, and implement than individual commands.
- The instrument performs more quickly and efficiently when it processes scripts than it does when it processes individual commands.
- You can incorporate features such as looping and branching into scripts.
- Scripts allow the controller to perform other tasks while the instrument is running a script, enabling some parallel operation.
- Scripts eliminate repeated data transfer times from the controller.

In the instrument, the Test Script Processor (TSP®) scripting engine processes and runs scripts.

This section describes how to create, load, modify, and run scripts.

What is a script?

A script is a collection of instrument control commands and programming statements. Scripts that you create are referred to as **user scripts**.

Your scripts can be interactive. Interactive scripts display messages on the front panel of the instrument that prompt the operator to enter parameters.

Runtime and nonvolatile memory storage of scripts

Scripts are loaded into the runtime environment of the instrument. From there, they can be stored in nonvolatile memory in the instrument.

The runtime environment is a collection of global variables, which include scripts, that the user has defined. A global variable can be used to store a value while the instrument is turned on. When you create a script, the instrument creates a global variable with the same name so that you can reference the script more conveniently. After scripts are loaded into the runtime environment, you can run and manage them from the front panel of the instrument or from a computer. Information in the runtime environment is lost when the instrument is turned off.

Nonvolatile memory is where information is stored even when the instrument is turned off. Save scripts to nonvolatile memory to save them even if the power is cycled. The scripts that are in nonvolatile memory are loaded into the runtime environment when the instrument is turned on.

Scripts are placed in the runtime environment at the following times:

- When they are run.
- When they are loaded over a remote command interface.
- When the instrument is turned on (if they are stored in nonvolatile memory).

For detail on the amount of available memory, see [Memory considerations for the runtime environment](#) (on page 13-50).

NOTE

If you make changes to a script in the runtime environment, the changes are lost when the instrument is turned off. To save the changes, you must save them to nonvolatile memory. See [Saving a script to nonvolatile memory](#) (on page 13-11).

What can be included in scripts?

Scripts can include combinations of Test Script Processor (TSP®) commands and Lua code. TSP commands instruct the instrument to do one thing and are described in the command reference (see [TSP commands](#) (on page 14-10)). Lua is a scripting language that is described in [Fundamentals of programming for TSP](#) (on page 13-16).

Working with scripts

This section describes the basics of working with scripts.

You can create and manage scripts from the front panel or over a remote interface. Scripts can be saved in the instrument, on a computer, or on a USB flash drive.

Tools for managing scripts

You can use any of the following tools to manage scripts:

- The front-panel menu options and USB flash drive. For information, refer to [Saving setups](#) (on page 4-92).
- Messages sent to the instrument. For information, see [Load a script by sending commands over a remote interface](#) (on page 13-9).
- Keithley Test Script Builder (TSB) software, which is available at tek.com/keithley (tek.com/products/keithley). For more information, see [Creating a new TSP project](#) (on page 13-43).
- Your own development tool or program.
- The front-panel interface options in the Scripts menu. For information, refer to the following sections.

Script rules

You can have as many scripts as needed in the instrument. The only limitation is the amount of memory available to the runtime environment.

When a script is loaded into the runtime environment, a global variable with the same name as the script is created to reference the script.

Important points regarding scripts:

- Each script must have a unique name.
- Script names must not contain spaces.
- Script names must be less than 256 characters.
- If you load a new script with the same name as an existing script, an error event message is generated. You must delete the existing script before you create a new script with the same name.
- If you revise a script and save it to the instrument with a new name, the previously loaded script remains in the instrument with the original name.
- You can save scripts to nonvolatile memory in the instrument. Saving a script to nonvolatile memory allows the instrument to be turned off without losing the script. See [Saving a script to nonvolatile memory](#) (on page 13-11).

Loading a script into the instrument

You can load scripts from the front-panel display by copying them from a USB flash drive. You can also load them over a remote interface using `loadscript` commands.

If a script on a USB flash drive is named `autoinstall.tsp`, the script is automatically copied to the list of internal scripts when the drive is inserted into the instrument.

Loading a script using a USB flash drive

After loading a script onto a USB flash drive, you can copy the script using options on the front-panel display.

To load a script using a USB flash drive:

1. Insert the USB flash drive into the USB port on the front panel.
2. Press the **MENU** key.
3. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.
4. In the USB Scripts list, select the script you want to copy from the USB flash drive.
5. Select **<**. The file is transferred to the instrument, and the corresponding file name is displayed in the Internal Scripts box.

Load a script by sending commands over a remote interface

To load a script over the remote interface, you can use the `loadscript` and `endscript` commands.

Normally, when the instrument receives a command, it runs the command immediately. When the instrument receives the `loadscript` command, the instrument starts collecting subsequent messages instead of running them immediately.

The `endscript` command tells the instrument to stop collecting messages. It then compiles the collection of messages into a script. The script is stored as a function. This script is loaded into the runtime environment — you need to save it to store it in the instrument.

To load a script:

Send the `loadscript` command with a script name. This tells the instrument to start collecting messages for the function named `testInfo`:

```
loadscript testInfo
```

Send the commands for the script; this example displays text on the USER swipe screen when the script is run:

```
display.settext(display.TEXT1, "Batch 233")
display.settext(display.TEXT2, "Test Information")
display.changescreen(display.SCREEN_USER_SWIPE)
```

Send the command that tells the instrument that the script is complete:

```
endscript
```

Run the script by sending the script name followed by `()`:

```
testInfo()
```

The USER swipe screen on the front panel is displayed and shows the text "Batch 233 Test Information" when you run this script.

To save the script to nonvolatile memory, send the command:

```
testInfo.save()
```

To load a script by sending commands:

1. Send the command `loadscript scriptName`, where *scriptName* is the name of the script. The name must be a legal Lua variable name.
2. Send the commands that need to be included in the script.
3. Send the command `endscript`.
4. You can now run the script. Send the script name followed by `()`. For more information, see [Running scripts using a remote interface](#) (on page 13-11).

Running scripts using the front-panel interface

To run a script from the Scripts menu:

1. Press the **MENU** key.
2. Under Scripts, select **Run**. The RUN SCRIPTS window is displayed.
3. From the Available Scripts list, select the script you want to run.
4. Select **Run Selected**.

To run a script from the home screen:

1. Press the **HOME** key.
2. Select the Script Indicator. The available scripts are displayed.
3. Select the script. A confirmation prompt is displayed.
4. Select **Yes** to run the script.

Running scripts using a remote interface

You can run any script using `scriptVar.run()`. Replace `scriptVar` with the name of a script that is in nonvolatile or run-time memory.

Saving a script to nonvolatile memory

You can save scripts to nonvolatile memory. To keep a script through a power cycle, you must save the script to nonvolatile memory.

To save a script to nonvolatile memory:

1. Create and load a script (see [Working with scripts](#) (on page 13-7)).
2. Send the command `scriptVar.save()`, where `scriptVar` is the name of the script.

Example: Save a user script to nonvolatile memory

```
test1.save()
```

Assume a script named `test1` has been loaded. `test1` is saved into nonvolatile memory.

Saving a script to a USB flash drive

You can save scripts to a USB flash drive.

To save a script to an external USB flash drive:

1. Load a script.
2. Insert a USB flash drive into the USB port on the front panel.
3. Send the command `scriptVar.save("/usb1/filename.tsp")`, where `scriptVar` is the variable referencing the script and `filename` is the name of the file.

Rename a script

To rename a script in the runtime environment:

1. Load the script into the runtime environment with a different name.
2. Delete the previous version of the script.

To rename a script in nonvolatile memory:

Send the commands:

```
scriptVar = script.load(file)
scriptVar.save()
```

Where:

- `scriptVar` is the name of variable that references the script
- `file` is the path and file name of the script file to load

For example, to load a script named `test8` from the USB flash drive and save it to nonvolatile memory, send the commands:

```
test8 = script.load("/usb1/test8.tsp")
test8.save()
```

NOTE

If the new name is the same as a name that is already used for a script, an event message is displayed, and the script is not saved.

Retrieve a user script from the instrument

You can review user scripts that are in the nonvolatile memory of the instrument and retrieve them.

To see a list of scripts from the front-panel interface:

1. Press the **MENU** key.
2. Under Scripts, select **Manage**. The MANAGE SCRIPTS window is displayed.

The scripts are listed in the Internal Scripts list. To see the contents of the script, you can copy them to a USB flash drive. You can read the scripts with a text editor. See [Saving a script to a USB flash drive](#) (on page 13-12).

To retrieve the content of a script, use `scriptVar.source`, where `scriptVar` is the name of the script you want to retrieve. For example, to retrieve a script named `contactTest`, you would send:

```
print(contactTest.source)
```

The command is returned as a single string. The `loadscript` and `endscript` keywords are not included.

Deleting a user script using a remote interface

Deleting a user script deletes the script from the instrument.

To delete a script from the instrument:

Send the command:

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

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

Example: Delete a user script

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

Delete a user script named `test8` from the instrument.

Power up script

The power up script runs automatically when the instrument is powered on. To create a power up script, save a new script and name it `autoexec`. The `autoexec` script is automatically saved to nonvolatile memory. See [Saving a script to nonvolatile memory](#) (on page 13-11).

NOTE

If an `autoexec` script already exists, you must delete it by sending the `script.delete("autoexec")` command. Performing a system reset does not delete the `autoexec` script.

To set up the power up script from the front panel:

1. Press the **MENU** key.
2. Under Scripts, select **Run**.
3. Select **Copy to Power Up**. A dialog box confirms that the script was copied.
4. Select **OK**.

To save the power up script using remote commands:

Send the command:

```
autoexec.save()
```

To delete the existing `autoexec` script, send the command:

```
script.delete("autoexec")
```

Commands that cannot be used in scripts

You cannot use the following commands as variables in scripts:

NOTE

There are some functions that resemble some of the strings below but are actually defined TSP functions. For example, [printbuffer\(\)](#) (on page 14-369) is a function you can use in scripts. If you are uncertain, check the [TSP command reference](#) (on page 14-1) to verify that the string is part of a defined function.

■ <code>abort</code>	■ <code>login</code>
■ <code>bit</code>	■ <code>logout</code>
■ <code>createconfigscript</code>	■ <code>node</code>

- endflash
- endscrip
- flash
- fs
- io
- loadscrip
- loadandruncr
- opc
- prevflash
- printbuffer
- printnumber
- scpi
- table
- waitcomplete

Common commands that cannot be used in scripts are shown in the following table with equivalent commands that can be used.

Unavailable commands with TSP equivalents	
Common commands	TSP equivalent commands
*CLS	eventlog.clear() status.clear()
*ESE	status.standard.enable
*ESE?	print(status.standard.enable)
*ESR?	print(status.standard.event)
*IDN?	print(localnode.model) print(localnode.serialno) print(localnode.version)
*LANG	No equivalent
*LANG?	No equivalent
*OPC	opc()
*OPC?	waitcomplete() print([[1]])
*RST	reset()
*SRE	status.request_enable
*SRE?	print(status.request_enable)
*STB?	print(status.condition)
*TRG	No equivalent
*TST?	print([[0]])
*WAI	waitcomplete()

Fundamentals of programming for TSP

To conduct a test, a computer (controller) is programmed to send sequences of commands to an instrument. The controller orchestrates the actions of the instrumentation. The controller is typically programmed to request measurement results from the instrumentation and make test sequence decisions based on those measurements.

To use the advanced features of the instrument, you can add programming commands to your scripts. Programming commands control script execution and provide tools such as variables, functions, branching, and loop control.

The Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands.

What is Lua?

Lua is a programming language that can be used with TSP-enabled instruments. Lua is an efficient language with simple syntax that is easy to learn.

Lua is also a scripting language, which means that scripts are compiled and run when they are sent to the instrument. You do not compile them before sending them to the instrument.

Lua basics

This section contains the basics about the Lua programming language to allow you to start adding Lua programming commands to your scripts quickly.

For more information about Lua, see the [Lua website \(lua.org\)](http://lua.org). Another source of useful information is the [Lua users group \(lua-users.org\)](http://lua-users.org), created for and by users of Lua programming language.

Comments

You can start a comment anywhere outside a string by typing a double hyphen (--). If the text immediately after -- is anything other than double left brackets ([[), the comment is a short comment, which continues until the end of the line.

If -- is followed by [[, the following characters are a long comment, which continues until double right brackets (]]) close the comment. Long comments may continue for several lines and may contain nested [[. . .]] pairs. The examples below show how to use code comments.

An example of a short comment is:

```
-- Turn off the front-panel display.
```

An example of a long comment is:

```
--[[Display a menu with three menu items. If the second menu item is selected,  
the selection is given the value Test2.]]
```

Function and variable name restrictions

You cannot use Lua reserved words and top-level command names for function or variable names.

Variable names must contain at least three characters.

The following table lists some of the Lua reserved words. If you attempt to assign these, the event code -285, "TSP Syntax error at line x: unexpected symbol near '*word*' " is displayed, where *word* is the Lua reserved word.

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

Values and variable types

In Lua, you use variables to store values in the runtime environment for later use.

Lua is a dynamically-typed language; the type of the variable is determined by the value that is assigned to the variable.

Variables in Lua are assumed to be global unless they are explicitly declared to be local. A global variable is accessible by all commands. Global variables do not exist until they have been assigned a value.

Variable types

Variables can be one of the following types.

Variable types and values

Variable type returned	Value	Notes
"nil"	not declared	The type of the value <code>nil</code> , whose main property is to be different from any other value; usually it represents the absence of a useful value.
"boolean"	true or false	Boolean is the type of the values <code>false</code> and <code>true</code> . In Lua, both <code>nil</code> and <code>false</code> make a condition <code>false</code> ; any other value makes it <code>true</code> .
"number"	number	All numbers are real numbers; there is no distinction between integers and floating-point numbers.
"string"	sequence of words or characters	
"function"	a block of code	Functions perform a task or compute and return values.
"table"	an array	New tables are created with <code>{ }</code> braces. For example: <code>{1, 2, 3.00e0}</code>
"userdata"	variables	Allows arbitrary program data to be stored in Lua variables.
"thread"	line of execution	

To determine the type of a variable, you can call the `type()` function, as shown in the examples below.

NOTE

The output you get from these examples may vary depending on the data format that is set.

Example: Nil

```
x = nil
print(x, type(x))
```

```
nil      nil
```

Example: Boolean

```
y = false
print(y, type(y))
```

```
false    boolean
```

Example: Hex constant

You can enter hexadecimal values, but to return a hexadecimal value, you must create a function, as shown in this example. Note that hexadecimal values are handled as a number type.

```
hex = function (i) return "0x"..string.format("%X", i) end
print(hex(0x54|0x55))
print(hex(0x54&0x66))
```

Set the format to return hexadecimal values, then OR two hexadecimal values and AND two hexadecimal values.

Output:

```
0x55
```

```
0x44
```

Example: Binary constant

Binary values are returned as floating point decimal values. Note that binary values are handled as a number type.

```
x = 0b0000000011111111
y = 0b1111111100000000
print(x, type(x))
print(y, type(y))
```

```
255 number
65280 number
```

Example: String and number

```
x = "123"
print(x, type(x))
```

```
123      string
```

```
x = x + 7
print(x, type(x))
```

```
Adding a number to x forces its type to number.
130      number
```

Example: Function

```
function add_two(first_value,
  second_value)
  return first_value + second_value
end
print(add_two(3, 4), type(add_two))
```

```
7        function
```

Example: Table

```
atable = {1, 2, 3, 4}
print(atable, type(atable))
print(atable[1])
print(atable[4])
```

Defines a table with four numeric elements. Note that the `table` value (shown here as `a096cd30`) varies.

```
table: a096cd30      table
1
4
```

Delete a global variable

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the runtime environment.

Operators

You can compare and manipulate Lua variables and constants using operators.

Arithmetic operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
-	negation (for example, <code>c = -a</code>)
^	exponentiation

Relational operators

Operator	Description
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
~=	not equal
!=	not equal
==	equal

Bitwise operators

Operator	Description
&	AND
	OR
^^	exclusive OR
<<	bitwise shift left
>>	bitwise shift right
!	logical NOT

Logical and bitwise operators

The logical operators in Lua are `and`, `or`, and `not`. All logical operators consider both `false` and `nil` as false and anything else as true.

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

The conjunction operator `and` returns its first argument if the first argument is `false` or `nil`; otherwise, `and` returns its second argument. The disjunction operator `or` returns its first argument if this value is different from `nil` and `false`; otherwise, `or` returns its second argument. Both `and` and `or` use shortcut evaluation, that is, the second operand is evaluated only if necessary.

NOTE

The example output you get may vary depending on the data format settings of the instrument.

Example 1

```
print(10 or eventlog.next())
print(nil or "a")
print(nil and 10)
print(false and eventlog.next())
print(false and nil)
print(false or nil)
print(10 and 20)
```

Output:

```
10
a
nil
false
false
nil
20
```

Example 2

```
hex = function (i) return "0x"..string.format("%X", i) end
print(hex(0x54 | 0x55))
print(hex(0x54 & 0x66))
```

Set the format to return hexadecimal values, then `OR` two hexadecimal values and `AND` two hexadecimal values.

Output:

```
0x55
0x44
```

Example 3

```
hex = function (i) return "0x"..string.format("%X", i) end
a, b = 0b01010100, 0b01100110
print(hex(a), "&", hex(b), "=", hex(a & b))
```

Set the format to return hexadecimal values, define binary values for a and b, then AND a and b.

Output:

0x54 & 0x66 = 0x44

String concatenation

String operators

Operator	Description
..	Concatenates two strings. If either argument is a number, it is coerced to a string (in a reasonable format) before concatenation.

Example: Concatenation

```
print(2 .. 3)
print("Hello " .. "World")
```

Output:

23
Hello World

Operator precedence

Operator precedence in Lua follows the order below (from higher to lower priority):

- ^ (exponentiation)
- not, - (unary), ! (logical NOT)
- *, /, <<, >>
- +, -, &, |, ^^
- .. (concatenation)
- <, >, <=, >=, ~=, !=, ==
- and
- or

You can use parentheses to change the precedences in an expression. The concatenation (". .") and exponentiation ("^") operators are right associative. All other binary operators are left associative. The examples below show equivalent expressions.

Equivalent expressions

<code>reading + offset < testValue/2+0.5</code>	<code>= (reading + offset) < ((testValue/2)+0.5)</code>
<code>3+reading^2*4</code>	<code>= 3+((reading^2)*4)</code>
<code>Rdg < maxRdg and lastRdg <= expectedRdg</code>	<code>= (Rdg < maxRdg) and (lastRdg <= expectedRdg)</code>
<code>-reading^2</code>	<code>= -(reading^2)</code>
<code>reading^testAdjustment^2</code>	<code>= reading^(testAdjustment^2)</code>

Functions

With Lua, you can group commands and statements using the `function` keyword.

Functions can take zero, one, or multiple parameters, and they return zero, one, or multiple values.

You can use functions to form expressions that calculate and return a value. Functions can also act as statements that execute specific tasks.

Functions are first-class values in Lua. That means that functions can be stored in variables, passed as arguments to other functions, and returned as results. They can also be stored in tables.

Note that when a function is defined, it is stored in the runtime environment. Like all data that is stored in the runtime environment, the function persists until it is removed from the runtime environment, is overwritten, or the instrument is turned off.

Create functions using the function keyword

Functions are created with a message or in Lua code in either of the following forms:

```
function myFunction(parameterX) functionBody end  
myFunction = function (parameterX) functionBody end
```

Where:

- *myFunction*: The name of the function.
- *parameterX*: Parameter names. To use multiple parameters, separate the names with commas.
- *functionBody*: The code that is executed when the function is called.

To execute a function, substitute appropriate values for *parameterX* and insert them into a message formatted as:

```
myFunction(valueForParameterX, valueForParameterY)
```

Where *valueForParameterX* and *valueForParameterY* represent the values to be passed to the function call for the given parameters.

NOTE

The output you get from these examples may vary depending on the data format settings of the instrument.

Example 1

```
function add_two(first_value,  
    second_value)  
    return first_value + second_value  
end  
print(add_two(3, 4))
```

Creates a variable named `add_two` that has a variable type of function.
Output:
7

Example 2

```
add_three = function(first_value,  
    second_value, third_value)  
    return first_value + second_value +  
        third_value  
end  
print(add_three(3, 4, 5))
```

Creates a variable named `add_three` that has a variable type of function.
Output:
12

Example 3

```
function sum_diff_ratio(first_value,  
    second_value)  
    psum = first_value + second_value  
    pdif = first_value - second_value  
    prat = first_value / second_value  
    return psum, pdif, prat  
end  
sum, diff, ratio = sum_diff_ratio(2, 3)  
print(sum)  
print(diff)  
print(ratio)
```

Returns multiple parameters (sum, difference, and ratio of the two numbers passed to it).

Output:

5

-1

0.666666666666667

Create functions using scripts

You can use scripts to define functions. Scripts that define a function are like any other script: They do not cause any action to be performed on the instrument until they are executed. The global variable of the function does not exist until the script that created the function is executed.

A script can consist of one or more functions. Once a script has been run, the computer can call functions that are in the script directly.

For detail on creating functions, see [Fundamentals of scripting for TSP](#) (on page 13-4).

Conditional branching

Lua uses the `if`, `else`, `elseif`, `then`, and `end` keywords to do conditional branching.

Note that in Lua, `nil` and `false` are false and everything else is true. Zero (0) is true in Lua.

The syntax of a conditional block is as follows:

```
if expression then
    block
elseif expression then
    block
else
    block
end
```

Where:

- *expression* is Lua code that evaluates to either true or false
- *block* consists of one or more Lua statements

Example: If

```
if 0 then
    print("Zero is true!")
else
    print("Zero is false.")
end
```

Output:
Zero is true!

Example: Comparison

```
x = 1
y = 2
if x and y then
    print("Both x and y are true")
end
```

Output:
Both x and y are true

Example: If and else

```
x = 2
if not x then
    print("This is from the if block")
else
    print("This is from the else block")
end
```

Output:
This is from the else block

Example: Else and elseif

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

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

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

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

Output:

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

Loop control

If you need to repeat code execution, you can use the Lua `while`, `repeat`, and `for` control structures. To exit a loop, you can use the `break` keyword.

While loops

To use conditional expressions to determine whether to execute or end a loop, you use `while` loops. These loops are similar to [Conditional branching](#) (on page 13-26) statements.

```
while expression do
    block
end
```

Where:

- *expression* is Lua code that evaluates to either `true` or `false`
- *block* consists of one or more Lua statements

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: While

```
list = {
    "One", "Two", "Three", "Four", "Five", "Six"}
print("Count list elements on numeric index:")
element = 1
while list[element] do
    print(element, list[element])
    element = element + 1
end
```

This loop exits when `list[element] = nil`.

Output:

Count list elements on
numeric index:

```
1  One
2  Two
3  Three
4  Four
5  Five
6  Six
```

Repeat until loops

To repeat a command, you use the `repeat ... until` statement. The body of a `repeat` statement always executes at least once. It stops repeating when the conditions of the `until` clause are met.

```
repeat
    block
until expression
```

Where:

- *block* consists of one or more Lua statements
- *expression* is Lua code that evaluates to either `true` or `false`

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Repeat until

```
list = {"One", "Two", "Three", "Four", "Five", "Six"}
print("Count elements in list using repeat:")
element = 1
repeat
    print(element, list[element])
    element = element + 1
until not list[element]
```

Output:

```
Count elements in list
using repeat:
1 One
2 Two
3 Three
4 Four
5 Five
6 Six
```

For loops

There are two variations of `for` statements supported in Lua: Numeric and generic.

NOTE

In a `for` loop, the loop expressions are evaluated once, before the loop starts.

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Numeric for

```
list = {"One", "Two", "Three", "Four", "Five", "Six"}
----- For loop -----
print("Counting from one to three:")
for element = 1, 3 do
    print(element, list[element])
end
print("Counting from one to four, in steps of two:")
for element = 1, 4, 2 do
    print(element, list[element])
end
```

The numeric `for` loop repeats a block of code while a control variable runs through an arithmetic progression.

Output:

```
Counting from one to three:
1   One
2   Two
3   Three
Counting from one to four, in steps of two:
1   One
3   Three
```

Example: Generic for

```

days = {"Sunday",
        "Monday",   "Tuesday",
        "Wednesday", "Thursday",
        "Friday",    "Saturday"}

for i, v in ipairs(days) do
    print(days[i], i, v)
end

```

The generic `for` statement works by using functions called iterators. On each iteration, the iterator function is called to produce a new value, stopping when this new value is nil.

Output:

Sunday	1	Sunday
Monday	2	Monday
Tuesday	3	Tuesday
Wednesday	4	Wednesday
Thursday	5	Thursday
Friday	6	Friday
Saturday	7	Saturday

Break

The `break` statement terminates the execution of a `while`, `repeat`, or `for` loop, skipping to the next statement after the loop. A `break` ends the innermost enclosing loop.

Return and `break` statements can only be written as the last statement of a block. If it is necessary to return or `break` in the middle of a block, an explicit inner block can be used.

NOTE

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Break with while statement

```

local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
local breakValue = 3
while k > 0 do
    if numTable[k] == breakValue then
        print("Going to break and k = ", k)
        break
    end
    k = k - 1
end
if k == 0 then
    print("Break value not found")
end

```

This example defines a `break` value (`breakValue`) so that the `break` statement is used to exit the `while` loop before the value of `k` reaches 0.

Output:

Going to break and k = 3

Example: Break with while statement enclosed by comment delimiters

```

local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
-- local breakValue = 3
while k > 0 do
    if numTable[k] == breakValue then
        print("Going to break and k = ", k)
        break
    end
    k = k - 1
end
if k == 0 then
    print("Break value not found")
end

```

This example defines a break value (breakValue), but the break value line is preceded by comment delimiters so that the break value is not assigned, and the code reaches the value 0 to exit the while loop.

Output:

Break value not found

Example: Break with infinite loop

```

a, b = 0, 1
while true do
    print(a, b)
    a, b = b, a + b
    if a > 500 then
        break
    end
end

```

This example uses a break statement that causes the while loop to exit if the value of a becomes greater than 500.

Output:

```

0      1
1      1
1      2
2      3
3      5
5      8
8      13
13     21
21     34
34     55
55     89
89     144
144    233
233    377
377    610

```

Tables and arrays

Lua makes extensive use of the data type table, which is a flexible array-like data type. Table indices start with 1. Tables can be indexed not only with numbers, but with any value except `nil`. Tables can be heterogeneous, which means that they can contain values of all types except `nil`.

Tables are the sole data structuring mechanism in Lua. They may be used to represent ordinary arrays, symbol tables, sets, records, graphs, trees, and so on. To represent records, Lua uses the field `name` as an index. The language supports this representation by providing `a.name` as an easier way to express `a["name"]`.

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Loop array

```
atable = {1, 2, 3, 4}
i = 1
while atable[i] do
    print(atable[i])
    i = i + 1
end
```

Defines a table with four numeric elements. Loops through the array and prints each element. The Boolean value of `atable[index]` evaluates to `true` if there is an element at that index. If there is no element at that index, `nil` is returned (`nil` is considered to be `false`).

Output:

```
1
2
3
4
```

Standard libraries

In addition to the standard programming constructs described in this document, Lua includes standard libraries that contain useful functions for string manipulation, mathematics, and related functions. Test Script Processor (TSP®) scripting engine instruments also include instrument control extension libraries, which provide programming interfaces to the instrumentation that can be accessed by the TSP scripting engine. These libraries are automatically loaded when the TSP scripting engine starts and do not need to be managed by the programmer.

The following topics provide information on some of the basic Lua standard libraries. For additional information, see the [Lua website](http://lua.org) (lua.org).

NOTE

When referring to the Lua website, please be aware that the TSP scripting engine uses Lua 5.0.2.

Base library functions

Base library functions

Function	Description
<code>collectgarbage()</code> <code>collectgarbage(<i>limit</i>)</code>	Sets the garbage-collection threshold to the given limit (in kilobytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, Lua immediately runs the garbage collector. If there is no limit parameter, it defaults to zero (0), which forces a garbage-collection cycle. See Lua memory management (on page 13-35) for more information.
<code>gcinfo()</code>	Returns the number of kilobytes of dynamic memory that the Test Script Processor (TSP®) scripting engine is using and returns the present garbage collector threshold (also in kilobytes). See Lua memory management (on page 13-35) for more information.
<code>tonumber(<i>x</i>)</code> <code>tonumber(<i>x</i>, <i>base</i>)</code>	Returns <i>x</i> converted to a number. If <i>x</i> is already a number, or a convertible string, the number is returned; otherwise, it returns <code>nil</code> . An optional argument specifies the base to use when interpreting the numeral. The base may be any integer from 2 to 36, inclusive. In bases above 10, the letter <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 and an optional exponent. In other bases, only unsigned integers are accepted.

Base library functions

Function	Description
<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 (as a string) the type of its only argument. The possible results of this function are "nil" (a string, not the value nil), "number", "string", "boolean", "table", "function", "thread", and "userdata".

Lua memory management

Lua automatically manages memory, which means you do not have to allocate memory for new objects and free it when the objects are no longer needed. Lua occasionally runs a garbage collector to collect all objects that are no longer accessible from Lua. All objects in Lua are subject to automatic management, including tables, variables, functions, threads, and strings.

Lua uses two numbers to control its garbage-collection cycles. One number counts how many bytes of dynamic memory Lua is using; the other is a threshold. When the number of bytes crosses the threshold, Lua runs the garbage collector, which reclaims the memory of all inaccessible objects. The byte counter is adjusted, and the threshold is reset to twice the new value of the byte counter.

String library functions

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

String library functions

Function	Description
<code>string.byte(s)</code> <code>string.byte(s, i)</code> <code>string.byte(s, i, j)</code>	Returns the internal numeric codes of the characters <code>s[i]</code> , <code>s[i+1]</code> , ..., <code>s[j]</code> . The default value for <code>i</code> is 1; the default value for <code>j</code> is <code>i</code> .
<code>string.char(...)</code>	Receives zero or more integers separated by commas. Returns a string with length equal to the number of arguments, in which each character has the internal numeric code equal to its corresponding argument.
<code>string.format(</code> <code>formatstring, ...)</code>	Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the <code>printf</code> family of standard C functions. The only differences are that the modifiers <code>*</code>, <code>l</code>, <code>L</code>, <code>n</code>, <code>p</code>, and <code>h</code> are not supported and there is an extra option, <code>q</code>. The <code>q</code> option formats a string in a form suitable to be safely read back by the Lua interpreter; the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written. For example, the call: <pre>string.format('%q', 'a string with "quotes" and \n new line')</pre> produces the string: <pre>"a string with \"quotes\" and \n new line"</pre> The options <code>c</code>, <code>d</code>, <code>E</code>, <code>e</code>, <code>f</code>, <code>g</code>, <code>G</code>, <code>i</code>, <code>o</code>, <code>u</code>, <code>X</code>, and <code>x</code> all expect a number as argument. <code>q</code> and <code>s</code> expect a string. This function does not accept string values containing embedded zeros, except as arguments to the <code>q</code> option.
<code>string.len(s)</code>	Receives a string and returns its length. The empty string <code>""</code> has length 0. Embedded zeros are counted, so <code>"a\000bc\000"</code> has length 5.
<code>string.lower(s)</code>	Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged.
<code>string.rep(s, n)</code>	Returns a string that is the concatenation of <code>n</code> copies of the string <code>s</code> .
<code>string.sub(s, i)</code> <code>string.sub(s, i, j)</code>	Returns the substring of <code>s</code> that starts at <code>i</code> and continues until <code>j</code> ; <code>i</code> and <code>j</code> can be negative. If <code>j</code> is absent, it is assumed to be equal to -1 (which is the same as the string length). In particular, the call <code>string.sub(s, 1, j)</code> returns a prefix of <code>s</code> with length <code>j</code> , and <code>string.sub(s, -i)</code> returns a suffix of <code>s</code> with length <code>i</code> .
<code>string.upper(s)</code>	Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged.

Math library functions

This library is an interface to most of the functions of the ANSI C math library. All trigonometric functions work in radians. The functions `math.deg()` and `math.rad()` convert between radians and degrees.

Math library functions

Function	Description
<code>math.abs(x)</code>	Returns the absolute value of x .
<code>math.acos(x)</code>	Returns the arc cosine of x .
<code>math.asin(x)</code>	Returns the arc sine of x .
<code>math.atan(x)</code>	Returns the arc tangent of x .
<code>math.atan2(y, x)</code>	Returns the arc tangent of y/x but uses the signs of both parameters to find the quadrant of the result (it also correctly handles the case of x being zero).
<code>math.ceil(x)</code>	Returns the smallest integer larger than or equal to x .
<code>math.cos(x)</code>	Returns the cosine of x .
<code>math.deg(x)</code>	Returns the angle x (given in radians) in degrees.
<code>math.exp(x)</code>	Returns the value e^x .
<code>math.floor(x)</code>	Returns the largest integer smaller than or equal to x .
<code>math.frexp(x)</code>	Returns m and e such that $x = m2^e$, where e is an integer and the absolute value of m is in the range $[0.5, 1]$ (or zero when x is zero).
<code>math.ldexp(m, e)</code>	Returns $m2^e$ (e should be an integer).
<code>math.log(x)</code>	Returns the natural logarithm of x .
<code>math.log10(x)</code>	Returns the base-10 logarithm of x .
<code>math.max(x, ...)</code>	Returns the maximum value among its arguments.
<code>math.min(x, ...)</code>	Returns the minimum value among its arguments.
<code>math.pi</code>	The value of π (3.141592654).
<code>math.pow(x, y)</code>	Returns x^y (you can also use the expression x^y to compute this value).
<code>math.rad(x)</code>	Returns the angle x (given in degrees) in radians.
<code>math.random()</code> <code>math.random(m)</code> <code>math.random(m, n)</code>	This function is an interface to the simple pseudorandom generator function <code>rand</code> provided by ANSI C. When called without arguments, returns a uniform pseudorandom real number in the range $[0, 1]$. When called with an integer number m , <code>math.random()</code> returns a uniform pseudorandom integer in the range $[1, m]$. When called with two integer numbers m and n , <code>math.random()</code> returns a uniform pseudorandom integer in the range $[m, n]$.
<code>math.randomseed(x)</code>	Sets x as the seed for the pseudorandom generator; equal seeds produce equal sequences of numbers.
<code>math.sin(x)</code>	Returns the sine of x .
<code>math.sqrt(x)</code>	Returns the square root of x . You can also use the expression $x^{0.5}$ to compute this value.
<code>math.tan(x)</code>	Returns the tangent of x .

Test Script Builder

Keithley Test Script Builder (TSB) is a software tool you can use to develop scripts for TSP-enabled instruments.

You must use the TSP command set to use TSB. Refer to [Determining the command set you will use](#) (on page 2-41) for information about the command sets and changing them.

Installing the TSB software

The installation files for the TSB software are available at tek.com/keithley (tek.com/products/keithley).

To install the TSB software:

1. Close all programs.
2. Download the installer to your computer and double-click the `.exe` file to start the installation.
3. Follow the on-screen instructions.

Using Test Script Builder (TSB)

Keithley Test Script Builder (TSB) is a software tool that simplifies building test scripts. You can use TSB to perform the following operations:

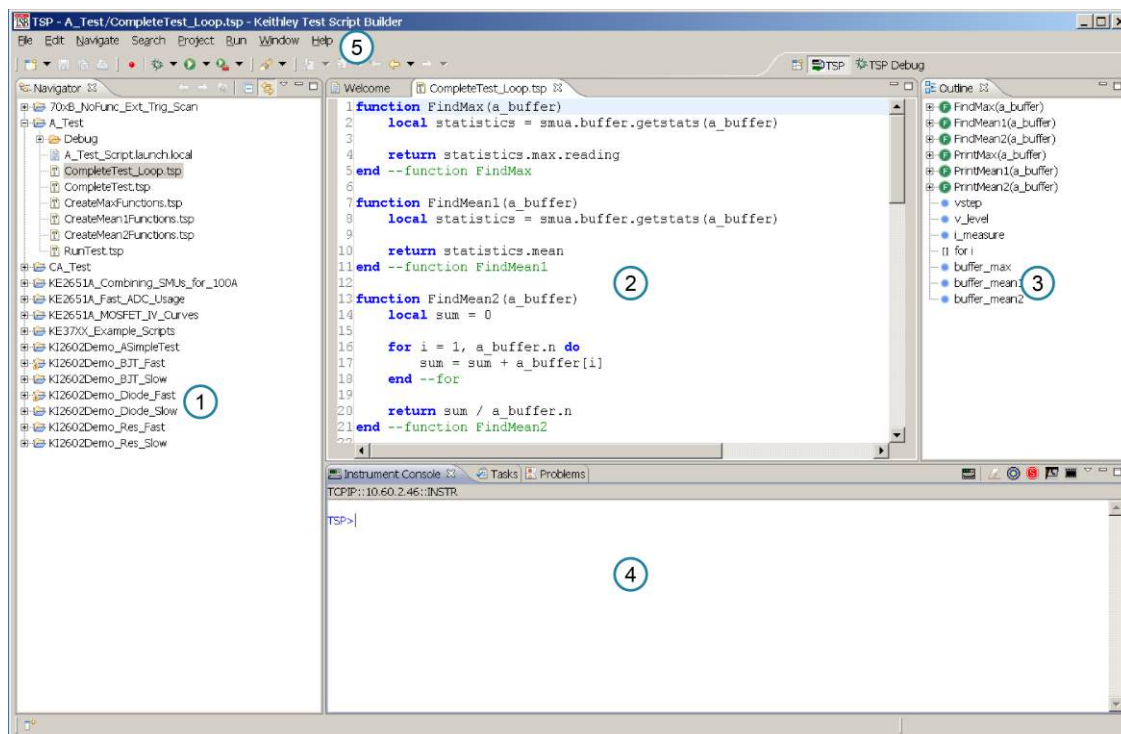
- Send remote commands and Lua statements
- Receive responses (data) from commands and scripts
- Upgrade instrument firmware
- Create, manage, and run user scripts
- Debug scripts
- Import factory scripts to view or edit and convert to user scripts

The Keithley Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands. For more information about using the Lua scripting language with Keithley TSP-enabled instruments, refer to the [Fundamentals of programming for TSP](#) (on page 13-16).

Keithley has created a collection of remote commands specifically for use with Keithley TSP-enabled instruments; for detailed information about those commands, refer to the "Command reference" section of the documentation for your specific instrument. You can build scripts from a combination of these commands and Lua programming statements. Scripts that you create are referred to as "user scripts." Also, some TSP-enabled instruments include built-in factory scripts.

The following figure shows an example of the Test Script Builder. As shown, the workspace is divided into these areas:

- Project navigator
- Script editor
- Outline view
- Programming interaction
- Help files

Figure 160: Example of the Test Script Builder workspace

Item	Description
1	Project navigator
2	Script editor; right-click to run the script that is displayed
3	Outline view
4	Programming interaction
5	Help; includes detailed information on using Test Script Builder

Project navigator

The project navigator consists of project folders and the script files (.tsp) created for each project. Each project folder can have one or more script files.

To view the script files in a project folder, select the plus (+) symbol next to the project folder. To hide the folder contents, select the minus (–) symbol next to the project folder.

You can download a TSP project to the instrument and run it, or you can run it from the TSB interface.

Script editor

The script editor is where you write, modify, and debug scripts.

To open and display a script file, double-click the file name in the project navigator. You can have multiple script files open in the script editor at the same time. Each open script file is displayed on a separate tab.

To display another script file that is already open, select the tab that contains the script in the script editor area.

Outline view

The outline view allows you to navigate through the structure of the active script in the script editor. Double-clicking a variable name or icon causes the first instance of the variable in the active script to be highlighted.

This view shows:

- Names of local and global variables
- Functions referenced by the active script in the script editor
- Parameters
- Loop control variables
- Table variables
- Simple assignments to table fields

Programming interaction

This part of the workspace is where you interact with the scripts that you are building in Test Script Builder (TSB). The actual contents of the programming interaction area of the workspace can vary.

You can send commands from the Instrument Console command line, retrieve data, view variables and errors, and view and set breakpoints when using the debug feature. For additional information, refer to the online help that is accessible from Test Script Builder (TSB).

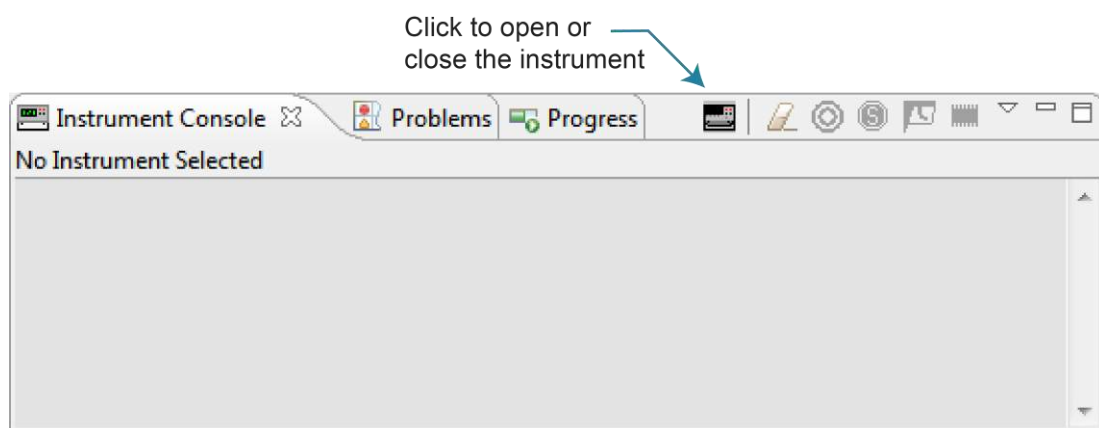
Connecting an instrument in TSB

You must use the TSP command set with the Test Script Builder software. For information on changing the command set, refer to [Determining the command set you will use](#) (on page 2-41).

To connect the Test Script Builder software to an instrument:

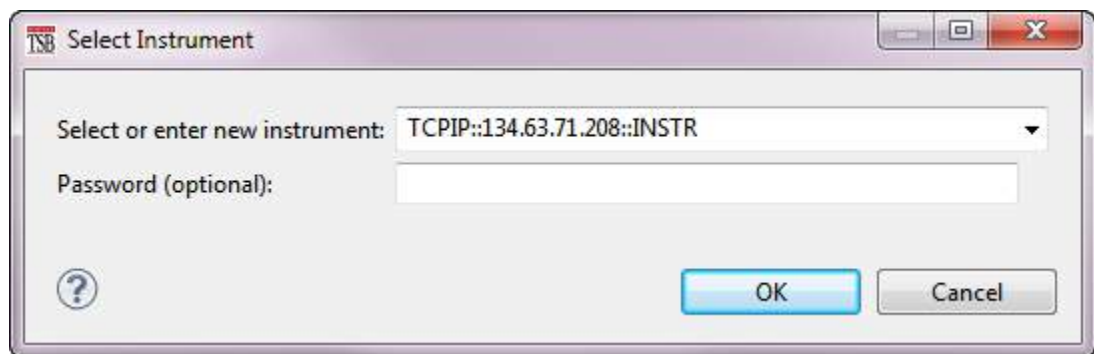
1. Select the **Open Instrument** icon in the script editor toolbar.

Figure 161: Opening an instrument connection in TSB



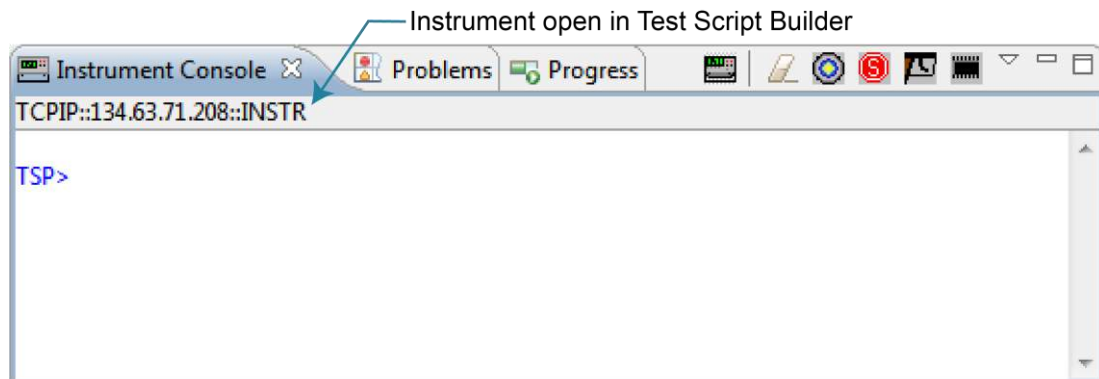
2. The Select Instrument dialog box opens. Select an existing instrument from the list or type the VISA resource ID of the instrument in the **Select or enter new instrument** box.
3. If needed, enter a password.

Figure 162: Select Instrument dialog box



4. Select **OK**. You briefly see the Opening Resource dialog box, and then the instrument is visible in the Instrument Console.

Figure 163: Instrument connected in TSB

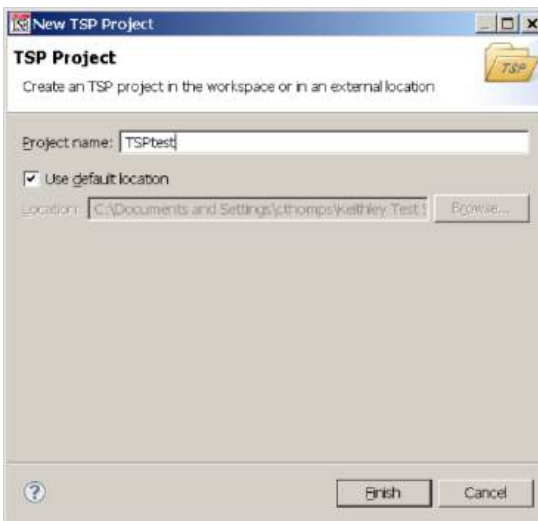


Creating a new TSP project

To create a new Test Script Processor (TSP®) project:

1. On the **File** menu in the TSP perspective, select **New > TSP Project**. The New TSP Project dialog box opens.

Figure 164: New TSP Project dialog box



2. Type a name for your project in the **Project name** box.
3. Select the location to create the new project.
4. Select **Finish**. The new project appears in the list of projects in the project navigator, and a file named `main.tsp` is created in the project. You can rename the `.tsp` file.
5. If you do not want to build your project automatically when it is saved or run, from the **Project** menu, clear **Build Automatically**.

NOTE

If you make changes to your project and do not build it before you run it, the Problems tab may not appear when problems are encountered.

Adding a new TSP file to a project

To add a new TSP file to a project:

1. Select the **File** menu and select **New > TSP File**. The New TSP File dialog box opens.
2. Select the project folder where you want to save the file.
3. Enter a name in the **File name** box.
4. Select **Finish**.

Running a script

You can run a script in the Test Script Builder (TSB) software using any of the following methods:

- Run a script that is open in the script editor area
- Run scripts that are listed in the Navigator area that are not currently open in the script editor window
- Run a collection of scripts by creating a run configuration (see [Creating a run configuration](#) (on page 13-45))

NOTE

When you use any of the run controls to run a script, the area that has focus in the workspace is important. For example, if the Navigator area is active (the tab is shaded) when you select the **Run** icon, the script file that is highlighted in the Navigator area is run instead of the active script in the script editor area.

The following list describes the most commonly used controls to run scripts in TSB:

- Right-click in the script editor area and select **Run Editor Contents** to run the active script as it currently appears in the script editor
- Right-click in the script editor area and select **Run As > 1 TSP File** to run the last saved version of the active script in the script editor as a .tsp file
- Select an action from the **Run** menu at the top of the TSB software interface

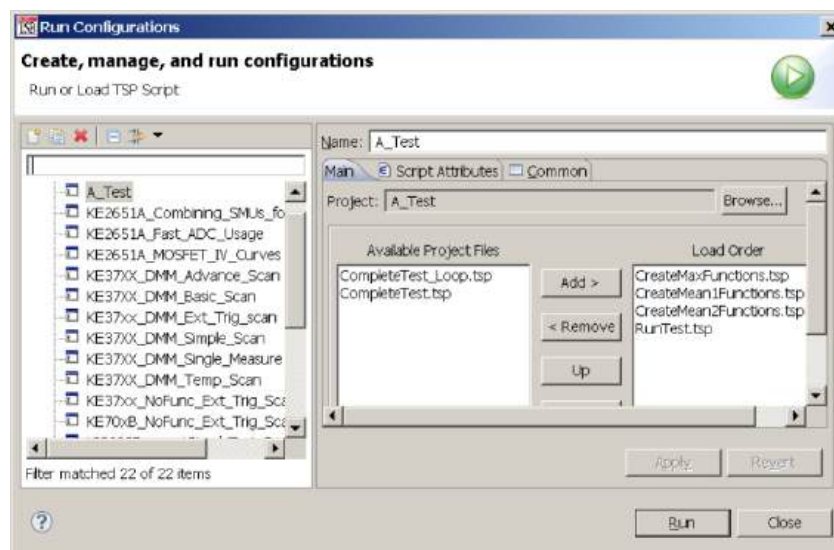
Creating a run configuration

A run configuration allows you to download multiple script files to an instrument and execute them as a single script.

To create a run configuration:

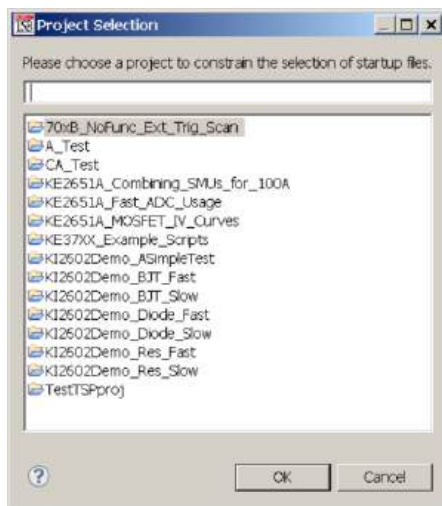
1. On the **Run** menu, select **Run Configurations**. The Run Configurations dialog opens.
2. The left pane of the dialog lists existing run and debug configurations. Select the script where the Run Configuration will be saved.
3. Select the **New launch configuration** icon  at the top left of the dialog. By default, a new configuration is created with the name `New_configuration`.

Figure 165: Run Configurations dialog box



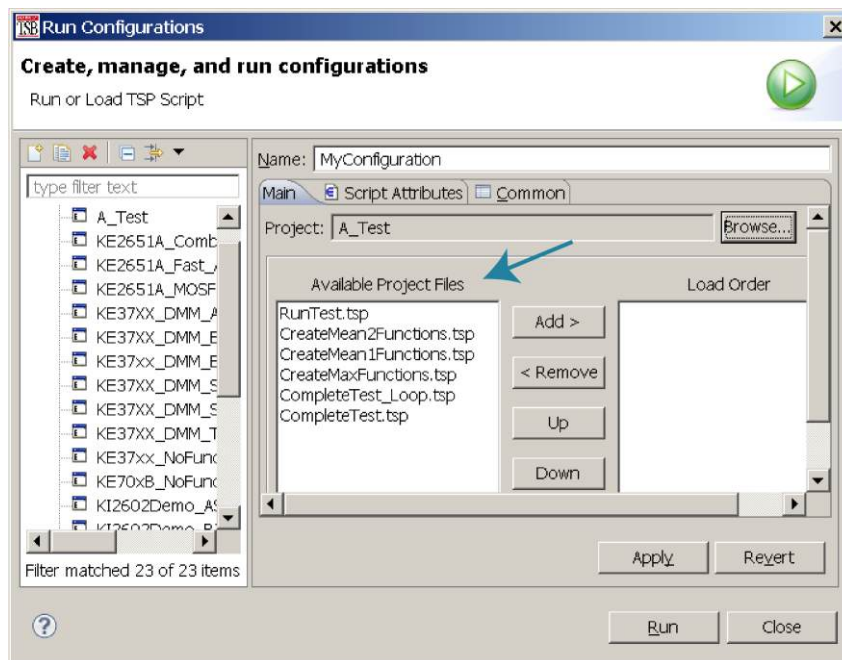
4. In the **Name** box, enter the name of your new run configuration.
5. Select the **Browse** button next the Project box.
6. Select a project from the list of available projects
7. Select **OK**.

Figure 166: Project Selection dialog box



The TSP files for the selected project are added to the Available Project Files list on the Main tab.

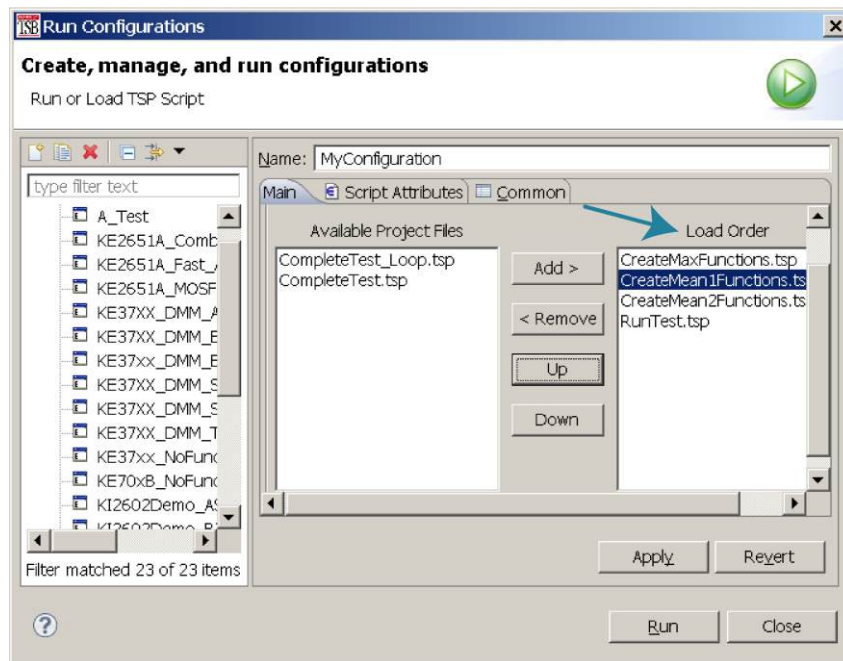
Figure 167: Available files for selected project



8. Select the files you want to add to the run configuration and select **Add** to add them to the Load Order list.

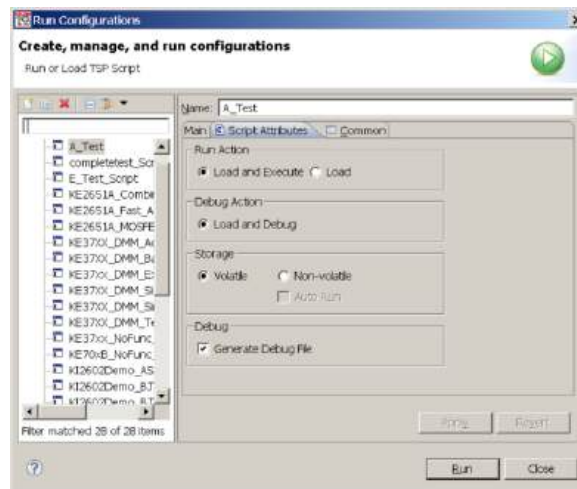
To change the load order of the TSP files, choose the files you want to move and select **Up** or **Down** until the files are in the correct order.

Figure 168: Selected TSP files load order



9. Select **Apply**.
10. Select the **Script Attributes** tab.
11. Select one of the following:
 - **Load and Execute:** If you select this option, which is the default selection, the script automatically loads into the instrument's volatile memory (runtime environment) and executes when you select **Run**.
 - **Load:** If you select this option, the script is loaded into the instrument's volatile memory when you select **Run** but is not executed until you manually run it. To manually run it from the command line in the Instrument Console, type `MyConfiguration.run()` (where *MyConfiguration* is the name of your configuration).

Figure 169: Script Attributes tab



12. In the Storage area of the Script Attributes tab, select **Volatile** or **Non-volatile**. For products that support autorun scripts, if you select Non-volatile, you can select **Auto Run** to have the script run automatically when the instrument is turned on. Note that all scripts are initially stored in the volatile (runtime) memory of the instrument memory and are lost if you turn the instrument power off and then on again. If you want to keep the script on the instrument through a power cycle, select **Non-volatile** storage.
13. In the Debug area of the Script Attributes tab, you can select **Generate Debug File**. When you select this option, a Debug subfolder is created in your test folder, and a file with a `.DBG` extension is created in that folder. Note that this is a feature of the Eclipse platform, and you will not use this file to debug your script. It contains all the scripts in your run configuration, so that you can see them together in the order in which they will load.
14. Select **Close** or **Run**. The run configuration is added to the run configurations list.

NOTE

To run the last used run configuration, select the **Run** icon  in the main TSB toolbar. To run a different run configuration, right-click in the script editor area and select **Run As > Run Configurations**. Select a different run configuration, and then select **Run** in the Run Configurations dialog.

Memory considerations for the runtime environment

The DAQ6510 reserves a large amount of memory for use with interactions with the front panel, commands, and test scripts. The amount of memory usage is affected by the following product features:

- Reading buffers (including local default and user-created reading buffers; if they are on a remote node, they only affect the remote node); refer to [Setting reading buffer capacity](#) (on page 6-11) for additional information.
- Scripts that are loaded onto the instrument.
- Applications that are loaded onto the instrument.
- Configuration lists.
- Scripts that are actively running.
- Variables that reside in the run-time environment.
- The TriggerFlow trigger model.
- USB drives that contain files that use a lot of memory, even if the files are not related to instrument operation.
- Scans.

The more a feature is used or the larger its definition, the more memory it consumes. For normal usage, reading buffers commonly reserve large amounts of memory. The amount of memory used depends on the number of readings and the buffer style.

CAUTION

The DAQ6510 notifies you when the system runs out of memory. If the instrument encounters memory allocation errors (errors that specifically state “Out of Memory”), the state of the instrument cannot be guaranteed. After attempting to save any important data, turn off power to the instrument and turn it back on to reset the runtime environment and return the instrument to a known state. Unsaved scripts and reading buffers will be lost.

If you encounter memory problems, examine the test script or SCPI commands that were being executed when the memory problems occurred. Take action to reduce the size of the elements that are consuming memory. If you are using TSP commands and scripting, also consider using the `collectgarbage()` command to clean up unused memory. For information on `collectgarbage()`, refer to [Base library functions](#) (on page 13-34).

The default size settings for the default reading buffers (`defbuffer1` and `defbuffer2`) are large. If your application does not use these buffers, you can set them to the minimum of 10 readings to conserve space. For information on adjusting the buffer size, refer to [Setting reading buffer capacity](#) (on page 6-11).

The buffer style is set when you create a user-defined reading buffer. The buffer style cannot be changed after the buffer has been created, so to eliminate memory problems caused by the style, you may need to delete or adjust the capacity of the buffers. Refer to [Creating buffers](#) (on page 6-6) for information on the effects of styles.

TSP command reference

In this section:

TSP command programming notes.....	14-1
Using the TSP command reference	14-5
TSP commands.....	14-10

TSP command programming notes

This section contains general information about using TSP commands.

TSP syntax rules

This section provides rules that explain what you can and cannot do when entering TSP commands.

Upper and lower case

Instrument commands are case sensitive.

Function and attribute names are in lowercase characters.

Parameters and attribute constants can use a combination of lowercase and uppercase characters. The correct case for a specific command is shown in its command description.

The following example shows the `beeper.beep()` function, where 2 is the duration in seconds and 2400 is the frequency. Note that the function is in lowercase characters:

```
beeper.beep(2, 2400)
```

The following command changes the display light state to be at level 50. Note that the attribute (`display.lightstate`) is lower case, but the constant (`display.STATE_LCD_50`) is a combination of lowercase and uppercase characters:

```
display.lightstate = display.STATE_LCD_50
```

White space

You can send commands with or without white spaces.

For example, the following functions, which set the length and frequency of the instrument beeper, are equivalent:

```
beeper.beep(2,2400)
beeper.beep (2, 2400)
```

Parameters for functions

All functions must have a set of parentheses () immediately following the function. If there are parameters for the function, they are placed between the parentheses. The parentheses are required even when no parameters are specified.

The following example shows the `beeper.beep()` function, where 2 is the duration in seconds and 2400 is the frequency. Note that the parameters are inside the parentheses:

```
beeper.beep(2, 2400)
```

The command below resets commands to their default values (no parameters are needed):

```
reset()
```

Multiple parameters

Multiple parameters must be separated by commas.

For example, the following commands set the beeper to emit a double-beep at 2400 Hz, with a beep sequence of 0.5 s on, a delay of 0.25 s, and then 0.5 s on:

```
beeper.beep(0.5, 2400)
delay(0.250)
beeper.beep(0.5, 2400)
```

Channel naming

In the DAQ6510, channels are named using the slot number and the channel number. The slot number is the number of the slot on the rear panel where the card is installed. The channel assignment is formatted as `SCH`, where:

- `S` is the slot number
- `CH` is a two-digit channel number

For example:

- `101` = slot 1, channel 1
- `210` = slot 2, channel 10

If you are sending commands from a remote interface, you include the three-digit channel name, `slot1`, `slot2`, or `allslots` in the channel list parameter for the commands.

In TSP commands, the channel numbers are in quotes. To designate multiple individual channels, separate the channels with commas. In the following example, the command opens channels 1 and 3 on the card in slot 1.

```
channel.open("101, 103")
```

To designate a range of channels, separate the first and last channel number with a colon. The range can be from lowest to highest numbered channel or highest to lowest. The following example sets channel 1 to 9 on slot 1 to the DC voltage measurement function and then creates a scan with those channels.

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)  
scan.create("101:109")
```

Some commands allow you to set all channels in a slot or all channels in the instrument. In this case, you can send `slot1` to set all channels in slot 1, `slot2` for all channels in slot 2, or `allslots` for all channels in all slots. For example, to set all channels on slot 1 to measure voltage, you can send:

```
channel.setdmm("allslots", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
```

When you send `slot1`, `slot2`, or `allslots` to a slot that contains channels that cannot accept that setting, the instrument generates an error, but the change is made to all channels for which the setting is allowed.

Time and date values

Time and date values are represented as the number of seconds since some base. The time bases are:

- **UTC 12:00 am Jan 1, 1970:** Some examples of UTC time are reading buffer timestamps, calibration adjustment and verification dates, and the value returned by `os.time()`.
- **Event:** Time referenced to an event, such as the first reading stored in a reading buffer.

Local and remote control

The instrument can be controlled locally or remotely.

When the instrument is controlled locally, you operate the instrument using the front-panel controls. When it is controlled remotely, you operate the instrument through a controller (usually a computer). When the instrument is first powered on, it is controlled locally.

The type of control is displayed on the Communications indicator of the display. When the instrument is in local control, Local is displayed.

When the instrument is in remote control, the type of control is displayed. Refer to [Communications indicator](#) (on page 3-10) front-panel for additional detail.

Remote control

When the instrument is controlled remotely, the front-panel controls are disabled. You can still view information on the front-panel display and move between the screens using the keys and touchscreen controls. If you change a selection, however, you are prompted to switch control to local.

To switch to remote control:

- Send a command from the computer to the instrument.
- Open communications between the instrument and Test Script Builder.

Local control

To change to local control, you can:

- Choose an option from the screens and try to change the value; select Yes on the dialog box that is displayed.
- Send the logout command from the computer.
- Turn the instrument off and on.

Using the TSP command reference

The TSP command reference contains detailed descriptions of each of the TSP commands that you can use to control your instrument. Each command description is broken into subsections. The figure below shows an example of a command description.

Figure 170: Example instrument command description

feature.enable

This command is an example of a typical TSP command that turns an instrument feature on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	feature.OFF

Usage

```
state = feature.enable
feature.enable = state
```

state

Disable the example feature: feature.OFF
Enable the example feature: feature.ON

Details

This command is an example of a typical TSP command that enables or disables a feature.

Example

feature.enable = feature.ON

Enables the example feature.

Also see

[exampleUnit.enable](#) (on page 1-73)

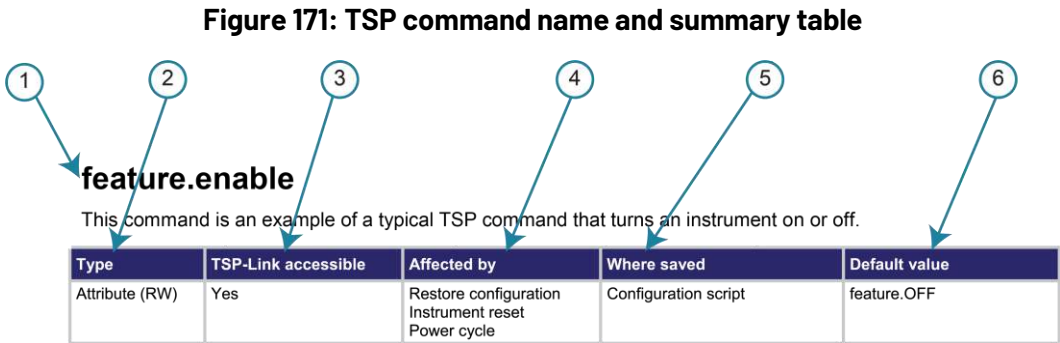
The subsections contain information about the command. The subsections are:

- Command name, brief description, and summary table
- Usage
- Details
- Example
- Also see

The content of each of these subsections is described in the following topics.

Command name, brief description, and summary table

Each instrument command description starts with the command name, followed by a brief description and a table with information for each command. Descriptions of the numbered items in the figure below are provided below.



- 1 Instrument command name.** The beginning of the command description. It is followed by a brief description of what the command does.
- 2 Type of command.** Commands can be functions, attributes, or constants. If the command is an attribute, it can be read-only (R), read-write (RW), or write-only (W).
- 3 TSP-Link accessible.** Indicates whether or not the command can be accessed through a TSP-Link network (Yes or No).

- 4 Affected by.** This column lists commands or actions that can change the value of the command, including:
- **Power cycle:** The command settings are not saved through a power cycle.
 - **Restore configuration:** If you restore a configuration script, this setting changes to the stored setting.
 - **Instrument reset:** When you reset the instrument, this command is reset to its default value. Reset can be done from the front panel or when you send `reset()` or `*RST`.
 - **Measure configuration list:** If you recall a measure configuration list, this setting changes to the setting stored in the list.
 - **Function:** This command changes value when the function is changed (for example, changing from a voltage measurement function to a current measurement function).
- 5 Where saved.** Indicates where the command settings reside once they are used on an instrument. Options include:
- **Not saved:** Command is not saved and must be typed each time you use it.
 - **Nonvolatile memory:** The command is stored in a storage area in the instrument where information is saved even when the instrument is turned off.
 - **Configuration script:** Command is saved as part of the configuration script.
 - **Measure configuration list:** This command is stored in measure configuration lists.
- 6 Default value:** Lists the default value or constant for the command.

Command usage

The Usage section of the remote command listing shows how to properly structure the command. Each line in the Usage section is a separate variation of the command usage. All possible command usage options are shown.

Figure 172: TSP usage description



- 1 Structure of command usage:** Shows how to organize the parts of the command. If a parameter is shown to the left of the command, it is the return when you print the command. Information to the right is the parameters or other items you need to enter when setting the command.
- 2 User-supplied parameters:** Indicated by italics. For example, for the function `beeper.beep(duration, frequency)`, replace *duration* with the number of seconds and *frequency* with the frequency of the tone. Send `beeper.beep(2, 2400)` to generate a two-second, 2400 Hz tone.

Some commands have optional parameters. If there are optional parameters, they must be entered in the order presented in the Usage section. You cannot leave out any parameters that precede the optional parameter. Optional parameters are shown as separate lines in usage, presented in the required order with each valid permutation of the optional parameters.

For example:

```
printbuffer(startIndex, endIndex, buffer1)  
printbuffer(startIndex, endIndex, buffer1, buffer2)
```

- 3 Parameter value options:** Descriptions of the options that are available for the user-defined parameter.

Command details

This section lists additional information you need to know to successfully use the remote command.

Figure 173: TSP Details description

Details
This command is a typical example of a command that enables or disables a feature.

Example section

The Example section of the remote command description shows examples of how you can use the command.

Figure 174: TSP example code



- 1 Actual example code that you can copy from this table and paste into your own programming application.
- 2 Description of the code and what it does. This may also contain example output of the code.

Related commands and information

The Also see section of the remote command description lists additional commands or sections that are related to the command.

Figure 175: TSP Also see description



TSP commands

The TSP commands available for the instrument are listed in alphabetical order.

available()

This function checks for the presence of specific instrument functionality.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
presence = available(functionality)
```

presence	If the functionality is present, returns <code>true</code> ; if not, returns <code>false</code>
functionality	The functionality to check for: ▪ Channel: <code>channel</code> ▪ Digital I/O: <code>digio</code> ▪ GPIB communications: <code>gpib</code> ▪ RS-232 communications: <code>serial</code> ▪ TSP-Link: <code>tsplink</code>

Details

The digital I/O, GPIB, RS-232, and TSP-Link options require an optional accessory card. This function allows you to check for these options using remote commands.

Example

```
print(available(gpib))
```

Returns `true` if GPIB communications are available. Returns `false` if GPIB communications are not available.

Also see

None

beeper.beep()

This function generates an audible tone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`beeper.beep(duration, frequency)`

<i>duration</i>	The amount of time to play the tone (0.001 s to 100 s)
<i>frequency</i>	The frequency of the beep (20 Hz to 8000 Hz)

Details

You can use the beeper of the instrument to provide an audible signal at a specific frequency and time duration.

Using this function from a remote interface does not affect audible errors or key click settings that were made from the DAQ6510 front panel.

Example

<code>beeper.beep(2, 2400)</code>	Generates a 2 s, 2400 Hz tone.

Also see

None

buffer.channelmath()

This function sets a math expression on a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_ADD,
    "channelNum2")
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_AVERAGE,
    "channelNum2")
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_DIVIDE,
    "channelNum2")
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_EXPONENT)
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_LOG10)
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_MULTIPLY,
    "channelNum2")
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_POLY,
    constant0, constant1, constant2, constant3, constant4, constant5)
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_POWER,
    constant0)
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_RATE)
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_RECIPROCAL)
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_SQROOT)
buffer.channelmath(readingBuffer, unit, "channelNumber", buffer.EXPR_SUBTRACT,
    "channelNum2")
```

<i>readingBuffer</i>	The name of the reading buffer; must be set to the style FULL

<i>unit</i>	The units to be applied to the value generated by the expression: ■ DC current: <code>buffer.UNIT_AMP</code> ■ AC current: <code>buffer.UNIT_AMP_AC</code> ■ Celsius: <code>buffer.UNIT_CELSIUS</code> ■ Custom unit 1 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM1</code> ■ Custom unit 2 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM2</code> ■ Custom unit 3 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM3</code> ■ DAC (voltage): <code>buffer.UNIT_DAC</code> ■ Decibel-milliwatts: <code>buffer.UNIT_DBM</code> ■ Decibels: <code>buffer.UNIT_DECIBEL</code> ■ Digital I/O: <code>buffer.UNIT_DIO</code> ■ Fahrenheit: <code>buffer.UNIT_FAHRENHEIT</code> ■ Capacitance: <code>buffer.UNIT_FARAD</code> ■ Frequency: <code>buffer.UNIT_HERTZ</code> ■ Kelvin: <code>buffer.UNIT_KELVIN</code> ■ No unit: <code>buffer.UNIT_NONE</code> ■ Resistance: <code>buffer.UNIT_OHM</code> ■ Percent: <code>buffer.UNIT_PERCENT</code> ■ DC voltage ratio: <code>buffer.UNIT_RATIO</code> ■ Reciprocal: <code>buffer.UNIT_RECIPROCAL</code> ■ Period: <code>buffer.UNIT_SECOND</code> ■ Totalizer: <code>buffer.UNIT_TOT</code> ■ DC voltage: <code>buffer.UNIT_VOLT</code> ■ AC voltage: <code>buffer.UNIT_VOLT_AC</code> ■ Power: <code>buffer.UNIT_WATT</code> ■ <code>buffer.UNIT_X</code>
<i>channelNumber</i>	String that contains the channel to apply the expression to, using normal channel list syntax; this is also the channel that supplies the <i>r</i> value in the expressions
<i>channelNum2</i>	String that contains the channel from which to get the previous measurement (the <i>a</i> value in the expressions); a measurement must be made on this channel before the <i>channelNumber</i>
<i>constant0</i>	The constant to be used for <i>c0</i> in the expression
<i>constant1</i>	The constant to be used for <i>c1</i> in the expression
<i>constant2</i>	The constant to be used for <i>c2</i> in the expression
<i>constant3</i>	The constant to be used for <i>c3</i> in the expression
<i>constant4</i>	The constant to be used for <i>c4</i> in the expression
<i>constant5</i>	The constant to be used for <i>c5</i> in the expression

Details

This command applies a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You can apply expressions to readings made from the DMM inputs or readings made from channels. If you have expressions set through both the buffer math and channel math commands, the expressions set for the channel math command take precedence.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style `FULL`. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

The expressions you can apply to readings are listed in the following table. In the formulas:

- r = present reading from the specified channel
- a = previous reading or reading from the specified channel
- t = timestamp of the reading
- c = constant

Expression	TSP parameter	Formula
Remove math expression	<code>buffer.EXPR_NONE</code>	Not applicable
Add	<code>buffer.EXPR_ADD</code>	$r + p$
Average	<code>buffer.EXPR_AVERAGE</code>	$\frac{(r+a)}{2}$
Divide	<code>buffer.EXPR_DIVIDE</code>	$\frac{r}{a}$
Exponent	<code>buffer.EXPR_EXPONENT</code>	10^r
Log10	<code>buffer.EXPR_LOG10</code>	$\log_{10} r$
Multiply	<code>buffer.EXPR_MULTIPLY</code>	$r * a$
Polynomial	<code>buffer.EXPR_POLY</code>	$c0 + c1 \cdot r + c2 \cdot r^2 + c3 \cdot r^3 + c4 \cdot r^4 + c5 \cdot r^5$
Power	<code>buffer.EXPR_POWER</code>	r^c
Rate of change	<code>buffer.EXPR_RATE</code>	$\frac{(r-r_1)}{(t-t_1)}$

Reciprocal	buffer.EXPR_RECIPROCAL	$\frac{1}{r}$
Square Root	buffer.EXPR_SQROOT	$\sqrt{r}$
Subtract	buffer.EXPR_SUBTRACT	$r - a$

Example

```

reset()
channelExp = buffer.make(200, buffer.STYLE_FULL)
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)

scan.create("101:105")
scan.scancount = 2
scan.buffer = channelExp
buffer.channelmath(channelExp, "102", buffer.UNIT_NONE, buffer.EXPR_SUBTRACT,
    "101")

trigger.model.initiate()
waitcomplete()
display.changescreen(display.SCREEN_READING_TABLE)
print(channelExp.extravalues[1])
printbuffer(1, 1, channelExp.extravalues)

```

This example assumes a 7700 switching module in slot 1.

Instrument has terminals set to REAR.

Reset the instrument.

Make a buffer named `channelExp` set to hold 200 readings with a buffer style of FULL.

Set the function for the channels 1 to 5 to voltage.

Set up a scan that includes channels 1 to 5.

Set the scan to use the reading buffer `channelExp`.

Set up channel math for channel 2, using a unit of none, that subtracts channel 1 from channel 2.

Start the scan.

Wait for results.

Read the data from channels 1 to 10, including the readings and the value generated by the expression.

Display the reading table on the front panel of the instrument.

Also see

[buffer.math\(\)](#) (on page 14-23)

[buffer.unit\(\)](#) (on page 14-32)

buffer.clearstats()

This function clears the statistical information associated with the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.clearstats()  
buffer.clearstats(bufferVar)
```

<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1 if not specified

Details

This command clears the statistics without clearing the readings.

Example

<code>buffer.clearstats()</code>	Clears statistics for defbuffer1.
<code>buffer.clearstats(testData)</code>	Clears statistics for testData.

Also see

[buffer.getstats\(\)](#) (on page 14-18)

buffer.delete()

This function deletes a user-defined reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.delete(bufferName)
```

<i>bufferName</i>	The name of a user-defined reading buffer

Details

You cannot delete the default reading buffers, `defbuffer1` and `defbuffer2`.
After deleting the buffer, call `collectgarbage()` to reclaim the memory the buffer was using.

Example

```
buf400 = buffer.make(400)
dmm.measure.read(buf400)
printbuffer(1, buf400.n, buf400.relativetimestamps)
buffer.delete(buf400)
collectgarbage()
```

Create a 400-element reading buffer named `buf400`.
Make measurements and store the readings in `buf400`.
Print the relative timestamps for each reading in the buffer.
Example output, assuming five readings are stored in the buffer:
`0, 0.412850017, 0.821640085, 1.230558058, 1.629523236`
Delete `buf400`.
Use `collectgarbage()` to unallocate the buffer.

Also see

- [buffer.make\(\)](#) (on page 14-22)
- [bufferVar.clear\(\)](#) (on page 14-43)
- [printbuffer\(\)](#) (on page 14-369)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

buffer.getstats()

This function returns statistics from a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
statsVar = buffer.getstats()
statsVar = buffer.getstats(bufferVar)
statsVar = buffer.getstats(bufferVar, "channelNumber")
statsVar = buffer.getstats(bufferVar, absStartTime, absStartFractional, absEndTime,
    absEndFractional)
statsVar = buffer.getstats(bufferVar, absStartTime, absStartFractional, absEndTime,
    absEndFractional, "channelNumber")
statsVar = buffer.getstats(bufferVar, relStartTime, relEndTime)
statsVar = buffer.getstats(bufferVar, relStartTime, relEndTime, "channelNumber")
```

<i>statsVar</i>	A table that contains the entries for buffer statistics; see Details for information on the entries
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; if no buffer is specified, this parameter defaults to defbuffer1
<i>channelNumber</i>	The channel number from which to retrieve data
<i>absStartTime</i>	An integer that represents the absolute start time in seconds
<i>absStartFractional</i>	An integer that represents the portion of the absolute start time that is in fractional seconds
<i>absEndTime</i>	An integer that represents the absolute end time in seconds
<i>absEndFractional</i>	An integer that represents the portion of the absolute end time that is in fractional seconds
<i>relStartTime</i>	The start time in seconds relative to the start time of the data in the buffer
<i>relEndTime</i>	The end time in seconds relative to the start time of the data in the buffer

Details

This function returns a table with statistical data about the data that was placed in the reading buffer.

The instrument automatically updates reading buffer statistics as data is added to the reading buffer.

When the reading buffer is configured to fill continuously and overwrite old data with new data, the buffer statistics include the data that was overwritten. To get statistics that do not include data that has been overwritten, define a large buffer size that will accommodate the number of readings you will make.

The table returned from this function provides statistics at the time the function is called. Although the instrument continues to update the statistics, the table that is returned is not updated. To get fresh statistics, call this function again.

The *statsVar* parameter contains the values described in the following table.

Attribute	When returned	Description
min	$n > 0$	A table that contains data about the minimum reading value that was added to the buffer; the table includes: - reading: The reading value - timestamp: The timestamp of the minimum data point in the buffer - seconds: The time in seconds - fractionalseconds: The time in fractional seconds
mean	$n > 0$	The average of all readings added to the buffer
stddev	$n > 1$	The standard deviation of all readings that were added to the buffer
n	Always	The number of data points on which the statistics are based
max	$n > 0$	A table that contains data about the maximum reading value that was added to the buffer; the table includes: - reading: The reading value - timestamp: The timestamp of the maximum data point in the buffer - seconds: The time in seconds - fractionalseconds: The fractional seconds

If *n* equals zero (0), all other values are *nil*. If *n* equals 1, *stddev* is *nil* because the standard deviation of a sample size of 1 is undefined.

Use the following command to get values from *statsVar*; a table with the following entries in it: *n*, *min*, *max*, *mean*, and *stddev*:

```
statsVar = buffer.getstats(bufferVar)
```

Use the following commands to print these entries:

```
print(statsVar.n)
print(statsVar.mean)
print(statsVar.stddev)
print(statsVar.min.reading)
print(statsVar.min.timestamp)
print(statsVar.min.seconds)
print(statsVar.min.fractionalseconds)
print(statsVar.max.reading)
print(statsVar.max.seconds)
print(statsVar.max.fractionalseconds)
print(statsVar.max.timestamp)
```

To print all the entries, send:

```
stats = buffer.getstats(defbuffer1)
print(stats)
```

Example 1

```
reset()
trigger.model.load("SimpleLoop", 12, 0.001, defbuffer1)
trigger.model.initiate()
waitcomplete()
```

```
stats = buffer.getstats(defbuffer1)
print(stats)
```

Reset the instrument.

Set up the SimpleLoop trigger-model template to make 12 readings with a 0.001 s delay. Readings are stored in defbuffer1.

Start the trigger model.

Assign the name `stats` to the table.

Get statistics for the default reading buffer named `defbuffer1`.

Example output:

```
["min"]={["seconds"]=1561123956, ["fractionalseconds"]=0.010184587,
  ["timestamp"]=1561123956, ["reading"]=8.4974199416e-05},
["mean"]=0.000132948335, ["stddev"]=4.4270141937e-05,
["max"]={["seconds"]=1561123955, ["fractionalseconds"]=0.833083981,
  ["timestamp"]=1561123955.8, ["reading"]=0.0002192359033}, ["n"]=12
```

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:105", dmm.ATTR_MEAS_NPLC, 0.01)
```

```
scan.create("101:105")
scan.scancount = 5
```

```
trigger.model.initiate()
waitcomplete()
```

```
stats = buffer.getstats(defbuffer1)
print(stats.min.reading)
print(stats.max.reading)
```

Set up channels 1 to 5 to measure DC voltage with an NPLC of 0.01.

Create a scan with these channels.

Set the scan count to 5 (make five measurements on each channel).

Start the scan.

Assign the name `stats` to the table.

Return the minimum and maximum readings from the buffer.

Example 3

```
-- This example shows how to use extra parameters, such as relative time.
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:105", dmm.ATTR_MEAS_NPLC, 0.01)
scan.create("101:105")
scan.scancount = 5
scan.scaninterval = 2
trigger.model.initiate()
waitcomplete()

stats = buffer.getstats(defbuffer1)
print("Minimum reading of all channels:", stats.min.reading)
print("Maximum reading of all channels:", stats.max.reading)
stats = buffer.getstats(defbuffer1, "103")
print("Minimum reading of all channel 103 readings:", stats.min.reading)
print("Maximum reading of all channel 103 readings:", stats.max.reading)
stats = buffer.getstats(defbuffer1, 103, 5)
print("Minimum reading of all channels from relative time 3 s to 5 s",
      stats.min.reading)
print("Maximum reading of all channels from relative time 3 s to 5 s",
      stats.max.reading)
```

Set up channels 1 to 5 to measure DC voltage with an NPLC of 0.01.

Create a scan with these channels.

Set the scan count to 5 (make five measurements on each channel).

Start the scan.

Assign the name `stats` to the table.

Return the minimum and maximum readings from the buffer.

Example output:

Minimum reading of all channels: -0.047647534452

Maximum reading of all channels: -0.015124015735

Minimum reading of all channel 3 readings: -0.04145936519

Maximum reading of all channel 3 readings: -0.025685636924

Minimum reading of all channels from relative time 3 s to 5 s -0.04623758547

Maximum reading of all channels from relative time 3 s to 5 s -0.019379596536

Also see

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

buffer.make()

This function creates a user-defined reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar = buffer.make(bufferSize)
bufferVar = buffer.make(bufferSize, style)
```

<i>bufferVar</i>	The name of the buffer
<i>bufferSize</i>	The maximum number of readings that can be stored in <i>bufferVar</i> ; minimum is 10; 0 to maximize buffer size (see Details)
<i>style</i>	The type of reading buffer to create: - Store readings with full accuracy with formatting: <code>buffer.STYLE_STANDARD</code> (default) - Store the same information as standard, plus additional information, such as the ratio component of a DCV ratio measurement: <code>buffer.STYLE_FULL</code> - Store external reading buffer data: <code>buffer.STYLE_WRITABLE</code> - Store external reading buffer data with two reading values: <code>buffer.STYLE_WRITABLE_FULL</code>

Details

You cannot assign user-defined reading buffers the name `defbuffer1` or `defbuffer2`. In addition, the buffer name must not already exist as a global variable, a local variable, table, or array.

If you create a reading buffer that has the same name as an existing user-defined buffer, the existing buffer is overwritten by the new buffer. Any data in the existing buffer is lost.

When you create a reading buffer, it becomes the active buffer. If you create two reading buffers, the last one you create becomes the active buffer.

If you select 0, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The default fill mode of a user-defined buffer is once. You can change it to continuous later.

Once the buffer style is selected, it cannot be changed.

Not all remote commands are compatible with the writable and full writable buffer styles. Check the Details section of the command descriptions before using them with any of these buffer styles.

Writable reading buffers are used to bring external data into the instrument. You cannot assign them to collect data from the instrument.

You can change the buffer capacity for an existing buffer through the front panel or by using the `bufferVar.capacity` command.

Example

```
capTest2 = buffer.make(200, buffer.STYLE_FULL)
Creates a 200-element reading buffer that stores readings with full accuracy named capTest2.
```

Also see

- [bufferVar.capacity](#) (on page 14-40)
- [bufferVar.fillmode](#) (on page 14-51)
- [buffer.write.format\(\)](#) (on page 14-34)
- [buffer.write.reading\(\)](#) (on page 14-37)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)
- [Writable reading buffers](#) (on page 6-36)

buffer.math()

This function allows you to run a mathematical expression on a measurement. The expression is applied when the measurement is placed in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.math(readingBuffer, unit, buffer.EXPR_ADD)
buffer.math(readingBuffer, unit, buffer.EXPR_AVERAGE)
buffer.math(readingBuffer, unit, buffer.EXPR_DIVIDE)
buffer.math(readingBuffer, unit, buffer.EXPR_EXPONENT)
buffer.math(readingBuffer, unit, buffer.EXPR_LOG10)
buffer.math(readingBuffer, unit, buffer.EXPR_MULTIPLY)
buffer.math(readingBuffer, unit, buffer.EXPR_NONE)
buffer.math(readingBuffer, unit, buffer.EXPR_POLY, constant0, constant1, constant2,
    constant3, constant4, constant5)
buffer.math(readingBuffer, unit, buffer.EXPR_POWER, constant0)
buffer.math(readingBuffer, unit, buffer.EXPR_RATE)
buffer.math(readingBuffer, unit, buffer.EXPR_RECIPROCAL)
buffer.math(readingBuffer, unit, buffer.EXPR_SQROOT)
buffer.math(readingBuffer, unit, buffer.EXPR_SUBTRACT)
```


<i>readingBuffer</i>	The name of the reading buffer; the reading buffer selected must be set to the style FULL
<i>unit</i>	The units to be applied to the value generated by the expression: ■ DC current: <code>buffer.UNIT_AMP</code> ■ AC current: <code>buffer.UNIT_AMP_AC</code> ■ Celsius: <code>buffer.UNIT_CELSIUS</code> ■ Custom unit 1 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM1</code> ■ Custom unit 2 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM2</code> ■ Custom unit 3 (defined by <code>buffer.unit()</code>): <code>buffer.UNIT_CUSTOM3</code> ■ DAC (voltage): <code>buffer.UNIT_DAC</code> ■ Decibel-milliwatts: <code>buffer.UNIT_DBM</code> ■ Decibels: <code>buffer.UNIT_DECIBEL</code> ■ Digital I/O: <code>buffer.UNIT_DIO</code> ■ Fahrenheit: <code>buffer.UNIT_FAHRENHEIT</code> ■ Capacitance: <code>buffer.UNIT_FARAD</code> ■ Frequency: <code>buffer.UNIT_HERTZ</code> ■ Kelvin: <code>buffer.UNIT_KELVIN</code> ■ No unit: <code>buffer.UNIT_NONE</code> ■ Resistance: <code>buffer.UNIT_OHM</code> ■ Percent: <code>buffer.UNIT_PERCENT</code> ■ DC voltage ratio: <code>buffer.UNIT_RATIO</code> ■ Reciprocal: <code>buffer.UNIT_RECIPROCAL</code> ■ Period: <code>buffer.UNIT_SECOND</code> ■ Totalizer: <code>buffer.UNIT_TOT</code> ■ DC voltage: <code>buffer.UNIT_VOLT</code> ■ AC voltage: <code>buffer.UNIT_VOLT_AC</code> ■ Power: <code>buffer.UNIT_WATT</code> ■ <code>buffer.UNIT_X</code>
<i>constant0</i>	The constant to be used for <code>c0</code> in the expression
<i>constant1</i>	The constant to be used for <code>c1</code> in the expression
<i>constant2</i>	The constant to be used for <code>c2</code> in the expression
<i>constant3</i>	The constant to be used for <code>c3</code> in the expression
<i>constant4</i>	The constant to be used for <code>c4</code> in the expression
<i>constant5</i>	The constant to be used for <code>c5</code> in the expression

Details

This command applies a mathematical expression to a reading as it is stored in the reading buffer. The result of the expression is then calculated and stored in the Extra column of the reading buffer.

You can apply expressions to readings made from the DMM inputs or readings made from channels. If you have expressions set through both the buffer math and channel math commands, the expressions set for the channel math command take precedence.

You must use remote commands to set up the expressions, but you can view results from the front panel using the reading table and the graph.

To use mathematical expressions, you must use a reading buffer that is set to the style `FULL`. You cannot use expressions with the default reading buffers (`defbuffer1` and `defbuffer2`).

The expressions you can apply to readings are listed in the following table. In the formulas:

- r = present reading from the specified channel
- a = previous reading or reading from the specified channel
- t = timestamp of the reading
- c = constant

Expression	TSP parameter	Formula
Remove math expression	<code>buffer.EXPR_NONE</code>	Not applicable
Add	<code>buffer.EXPR_ADD</code>	$r + p$
Average	<code>buffer.EXPR_AVERAGE</code>	$\frac{(r+a)}{2}$
Divide	<code>buffer.EXPR_DIVIDE</code>	$\frac{r}{a}$
Exponent	<code>buffer.EXPR_EXPONENT</code>	10^r
Log10	<code>buffer.EXPR_LOG10</code>	$\log_{10} r$
Multiply	<code>buffer.EXPR_MULTIPLY</code>	$r * a$
Polynomial	<code>buffer.EXPR_POLY</code>	$c0 + c1 \cdot r + c2 \cdot r^2 + c3 \cdot r^3 + c4 \cdot r^4 + c5 \cdot r^5$
Power	<code>buffer.EXPR_POWER</code>	r^c
Rate of change	<code>buffer.EXPR_RATE</code>	$\frac{(r-r_1)}{(t-t_1)}$
Reciprocal	<code>buffer.EXPR_RECIPROCAL</code>	$\frac{1}{r}$
Square Root	<code>buffer.EXPR_SQROOT</code>	$\sqrt{r}$
Subtract	<code>buffer.EXPR_SUBTRACT</code>	$r - a$

Example

```
reset()
mathExp = buffer.make(200, buffer.STYLE_FULL)
dmm.measure.func = dmm.FUNC_DC_VOLTAGE

buffer.math(mathExp, buffer.UNIT_NONE, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
    print("Reading: ", dmm.measure.read(mathExp))
end

display.changescreen(display.SCREEN_READING_TABLE)

print("Extra value reading 1: ", mathExp.extravalues[1])
print("Extra value reading 2: ", mathExp.extravalues[2])
print("Extra value reading 3: ", mathExp.extravalues[3])
```

Reset the instrument.

Make a buffer named `mathExp` set to hold 200 readings with a buffer style of `FULL`.

Set the measure function to DC voltage.

Set the buffer math expression to multiply readings against the previous readings.

Make three readings.

Display the reading table on the front panel of the instrument, where you can view the extra readings.

Print the extra values (the calculated values).

Example output:

Reading: 6.3863430578e-05

Reading: 6.7818055872e-05

Reading: 1.9871571784e-05

Extra value reading 1: 6.3863430578e-05

Extra value reading 2: 4.3310937031e-09

Extra value reading 3: 1.3476513655e-09

Also see

[buffer.channelmath\(\)](#) (on page 14-12)

[buffer.unit\(\)](#) (on page 14-32)

buffer.save()

This function saves data from the specified reading buffer to a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.save(bufferVar, "fileName")
buffer.save(bufferVar, "fileName", what)
buffer.save(bufferVar, "fileName", what, start, end)
```

<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>fileName</i>	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
<i>what</i>	Defines which information is saved in the file on the USB flash drive: see Details
<i>start</i>	Defines the starting point in the buffer to start saving data
<i>end</i>	Defines the ending point in the buffer to stop saving data

Details

The file name must specify the full path (including /usb1/). If included, the file extension must be set to .csv. If no file extension is specified, .csv is added.

Examples of valid destination file names:

```
buffer.save(MyBuffer, "/usb1/myData")
buffer.save(MyBuffer, "/usb1/myData.csv")
```

The DAQ6510 does not check for existing files when you save. Verify that you are using a unique name to avoid overwriting any existing .csv files on the flash drive.

Option	Export includes
buffer.COL_ALL	All data
buffer.COL_BRIEF	Reading and relative time
buffer.COL_CHANNEL	Channel
buffer.COL_CSV_CHAN_COLS	Ignore other columns and use special format with a column per channel
buffer.COL_CSV_EASY_GRAPH	Ignore other columns and use special format that is easy to graph in Microsoft Excel
buffer.COL_DISPLAY_DIGITS	The setting for the display digits
buffer.COL_EXTRA	Relative time and additional values if they exist (such as the sense voltage from a DC voltage ratio measurement)
buffer.COL_EXTRA_RANGE	Extra value range digits
buffer.COL_EXTRA_UNIT	Extra value units
buffer.COL_EXTRA_VALUE	Extra value
buffer.COL_INDEX	Index into buffer
buffer.COL_LIMITS	The status of all limits
buffer.COL_MATH	Math enabled (F if math is not enabled; T if math is enabled) and relative time
buffer.COL_ORIGIN	Origin status
buffer.COL_QUESTIONABLE	Questionable status
buffer.COL_RANGE_DIGITS	Range digits
buffer.COL_READING	The measurement reading
buffer.COL_STANDARD	The relative time, reading, channel, and source value
buffer.COL_START	Status of start group
buffer.COL_STATUS	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table in bufferVar.statuses (on page 14-61)
buffer.COL_TERMINAL	Terminal status
buffer.COL_TIME_ABSOLUTE	The time when the data point was measured as an absolute timestamp
buffer.COL_TIME_PARTS	Absolute time in multiple columns
buffer.COL_TIME_RAW	Absolute time in seconds
buffer.COL_TIME_RELATIVE	The relative time when the data point was measured in seconds
buffer.COL_TIMESTAMP_READING	The timestamp reading
buffer.COL_UNIT	The reading and the unit of measure

Example 1

```
buffer.save(MyBuffer, "/usb1/myData.csv")
```

Save all reading and default time information from a buffer named `MyBuffer` to a file named `myData.csv` on the USB flash drive.

Example 2

```
buffer.save(MyBuffer, "/usb1/myDataRel.csv", buffer.SAVE_RELATIVE_TIME)
```

Save all readings and relative timestamps from `MyBuffer` to a file named `myDataRel.csv` on the USB flash drive.

Example 3

```
buffer.save(defbuffer1, "/usb1/defbuf1data", buffer.SAVE_RAW_TIME)
```

Save readings and raw timestamps from `defbuffer1` to a file named `defbuf1data` on the USB flash drive.

Also see

[buffer.make\(\)](#) (on page 14-22)

[buffer.saveappend\(\)](#) (on page 14-29)

[Reading buffers](#) (on page 6-1, on page 6-1)

[Remote buffer operation](#) (on page 6-30)

buffer.saveappend()

This function appends data from the reading buffer to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.saveappend(bufferVar, "filename")
buffer.saveappend(bufferVar, "filename", timeFormat)
buffer.saveappend(bufferVar, "filename", timeFormat, start, end)
```

<i>bufferVar</i>	Indicates the reading buffer to use; the default buffers (<code>defbuffer1</code> or <code>defbuffer2</code>) or the name of a user-defined buffer; if no buffer is specified, <code>defbuffer1</code> is used
<i>fileName</i>	A string that indicates the name of the file on the USB flash drive in which to save the reading buffer
<i>timeFormat</i>	Indicates how date and time information from the buffer is saved in the file on the USB flash drive; the options are: - Save dates, times, and fractional seconds: <code>buffer.SAVE_FORMAT_TIME</code> - Save relative timestamps: <code>buffer.SAVE_RELATIVE_TIME</code> - Save seconds and fractional seconds: <code>buffer.SAVE_RAW_TIME</code> - Save timestamps: <code>buffer.SAVE_TIMESTAMP_TIME</code>
<i>start</i>	Defines the starting point in the buffer to start saving data
<i>end</i>	Defines the ending point in the buffer to stop saving data

Details

If the file you specify does not exist on the USB flash drive, this command creates the file.

For options that save more than one item of time information, each item is comma-delimited. For example, the default format is date, time, and fractional seconds for each reading.

The file extension `.csv` is appended to the file name if necessary. Any file extension other than `.csv` generates an error.

The index column entry in the `.csv` file starts at 1 for each append operation.

Examples of valid destination file names:

```
buffer.saveappend(bufferVar, "/usb1/myData")
buffer.saveappend(bufferVar, "/usb1/myData.csv")
```

Invalid destination file name examples:

```
buffer.saveappend(bufferVar, "/usb1/myData.")
```

— The period is not followed by `csv`.

```
buffer.saveappend(bufferVar, "/usb1/myData.txt")
```

— The only allowed extension is `.csv`. If `.csv` is not assigned, it is automatically added.

Example 1

```
buffer.saveappend(MyBuffer, "/usb1/myData.csv")
```

Append reading and default time information from a buffer named `MyBuffer` to a file named `myData.csv` on the USB flash drive.

Example 2

```
buffer.saveappend(MyBuffer, "/usb1/myDataRel.csv", buffer.SAVE_RELATIVE_TIME)
```

Append readings and relative timestamps from `MyBuffer` to a file named `myDataRel.csv` on the USB flash drive.

Example 3

```

reset()
if file.usbdriveexists() == 1 then
    testDir = "TestData11"
    -- Create a directory on the USB drive for the data.
    file.mkdir(testDir)
    -- Build the full file and path.
    fileName = "/usb1/" .. testDir .. "/myTestData.csv"
    -- Open the file where the data will be stored.
    fileNumber = file.open(fileName, file.MODE_WRITE)
    -- Write the string data to a file.
    file.write(fileNumber, "Tested to Company Standard ABC.101\n")
    -- Write the header separator to a file.
    file.write(fileNumber,
        "=====\n")
    -- Write the string data to a file.
    file.write(fileNumber, "\t1. Connect HI/LO to respective DUT terminals.\n")
    file.write(fileNumber, "\t2. Set power supply to 5 VDC @ 1 A.\n")
    file.write(fileNumber, "\t3. Wait 30 minutes.\n")
    file.write(fileNumber, "\t4. Capture 100 readings and analyze data.\n\n\n")
    -- Write buffering data to a file.
    file.flush(fileNumber)
    -- Close the data file.
    file.close(fileNumber)
end
-- Fix the range to 10 V.
dmm.measure.range = 10.0
-- Set the measurement count to 100.
dmm.measure.count = 100
-- Set up reading buffers.
-- Ensure the default measurement buffer size matches the count.
defbuffer1.capacity = dmm.measure.count
dmm.measure.read()
buffer.saveappend(defbuffer1, fileName)

```

Write string data to a file with information about a test file.

Also see

[buffer.make\(\)](#) (on page 14-22)

[buffer.save\(\)](#) (on page 14-27)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

buffer.unit()

This function allows you to create up to three custom units of measure for use in buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.unit(buffer.UNIT_CUSTOMN, unitOfMeasure)
```

<i>N</i>	The number of the custom unit, 1, 2, or 3
<i>unitOfMeasure</i>	A string that defines the custom unit; up to three characters; defaults are x for custom unit 1, y for unit 2, and z for unit 3

Details

You can use custom units of measures in buffer math, channel math, and writable buffers.

If you specify more than two characters, the additional characters are ignored. Some characters are converted to other symbols:

- u is displayed as μ .
- dC is displayed as $^{\circ}\text{C}$.
- dF is displayed as $^{\circ}\text{F}$.
- RA is displayed as V/V .

This unit is reset when power is cycled. It is not affected by reset.

Example

```
reset()
mathExp = buffer.make(200, buffer.STYLE_FULL)
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
buffer.unit(buffer.UNIT_CUSTOM1, "fb")

buffer.math(mathExp, buffer.UNIT_CUSTOM1, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
    print("Reading "..x..":", dmm.measure.read(mathExp))
end

display.changescreen(display.SCREEN_READING_TABLE)
for x = 1, 3 do
    print("Extra value reading "..x..":", mathExp.extravalues[x])
end
```

This example assumes a 7700 switching module in slot 1. Instrument has terminals set to FRONT.
Reset the instrument.
Make a buffer named `mathExp` set to hold 200 readings with a buffer style of FULL.
Set the measure function to DC voltage.
Set the customer 1 buffer unit to `fb`.
Set the buffer math expression to multiply readings against the previous readings.
Make 3 readings.
Display the reading table on the front panel of the instrument, where you can view the extra readings.
Print the extra values (the calculated values).
Example output:
Reading 1: 0.00015611271869
Reading 2: 9.0539004907e-05
Reading 3: 0.30001141669554
Extra value reading 1: 0.00015611271869
Extra value reading 2: 1.4134290203e-08
Extra value reading 3: 1.0336562635e-08

Also see

[buffer.channelmath\(\)](#) (on page 14-12)

[buffer.math\(\)](#) (on page 14-23)

[buffer.write.format\(\)](#) (on page 14-34)

buffer.write.format()

This function sets the units and number of digits of the readings that are written into the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buffer.write.format(bufferVar, units, displayDigits)
buffer.write.format(bufferVar, units, displayDigits, extraUnits)
buffer.write.format(bufferVar, units, displayDigits, extraUnits, extraDigits)
```

<i>bufferVar</i>	The name of the buffer
<i>units</i>	The units for the first measurement in the buffer index: ■ <code>buffer.UNIT_AMP</code> ■ <code>buffer.UNIT_AMP_AC</code> ■ <code>buffer.UNIT_CELSIUS</code> ■ <code>buffer.UNIT_CUSTOM1</code> ■ <code>buffer.UNIT_CUSTOM2</code> ■ <code>buffer.UNIT_CUSTOM3</code> ■ <code>buffer.UNIT_DAC</code> ■ <code>buffer.UNIT_DBM</code> ■ <code>buffer.UNIT_DECIBEL</code> ■ <code>buffer.UNIT_DIO</code> ■ <code>buffer.UNIT_FAHRENHEIT</code> ■ <code>buffer.UNIT_FARAD</code> ■ <code>buffer.UNIT_HERTZ</code> ■ <code>buffer.UNIT_KELVIN</code> ■ <code>buffer.UNIT_NONE</code> ■ <code>buffer.UNIT_OHM</code> ■ <code>buffer.UNIT_PERCENT</code> ■ <code>buffer.UNIT_RATIO</code> ■ <code>buffer.UNIT_RECIPROCAL</code> ■ <code>buffer.UNIT_SECOND</code> ■ <code>buffer.UNIT_TOT</code> ■ <code>buffer.UNIT_VOLT</code> ■ <code>buffer.UNIT_VOLT_AC</code> ■ <code>buffer.UNIT_WATT</code> ■ <code>buffer.UNIT_X</code>
<i>displayDigits</i>	The number of digits to use for the first measurement: ■ <code>buffer.DIGITS_3_5</code> ■ <code>buffer.DIGITS_4_5</code> ■ <code>buffer.DIGITS_5_5</code> ■ <code>buffer.DIGITS_6_5</code> ■ <code>buffer.DIGITS_7_5</code> ■ <code>buffer.DIGITS_8_5</code>
<i>extraUnits</i>	The units for the second measurement in the buffer index; the selections are the same as <i>units</i> (only valid for buffer style <code>WRITABLE_FULL</code>); if not specified, uses the value for <i>units</i>
<i>extraDigits</i>	The number of digits to use for the second measurement; the selections are the same as <i>displayDigits</i> (only valid for buffer style <code>WRITABLE_FULL</code>); if not specified, uses the value for <i>displayDigits</i>

Details

This command is valid when the buffer style is writable or full writable. When the buffer style is set to full writable, you can include an extra value.

The format defines the units and the number of digits that are reported for the data. This command affects how the data is shown in the reading buffer and what is shown on the front-panel Home, Histogram, Reading Table, and Graph screens.

Example 1

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1)
buffer.write.reading(extBuffer, 2)
buffer.write.reading(extBuffer, 3)
buffer.write.reading(extBuffer, 4)
buffer.write.reading(extBuffer, 5)
buffer.write.reading(extBuffer, 6)
printbuffer(1, 6, extBuffer.readings, extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is writable.

Set the data format to show units of watts with 3½ digit resolution.

Write 6 pieces of data into the buffer.

Print the buffer, including the readings and units.

Read the buffer.

Output:

```
1.0000000000e+00, Watt DC, 2.0000000000e+00, Watt DC, 3.0000000000e+00, Watt DC,
4.0000000000e+00, Watt DC, 5.0000000000e+00, Watt DC, 6.0000000000e+00, Watt
DC
```

Example 2

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
    buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
    extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is full writable.

Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.

Write 12 pieces of data into the buffer.

Print the buffer, including the readings and units.

Read the buffer.

Output:

```
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt
DC, 10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC
```

Also see

[buffer.make\(\)](#) (on page 14-22)

[buffer.unit\(\)](#) (on page 14-32)

[buffer.write.reading\(\)](#) (on page 14-37)

[Reading buffers](#) (on page 6-1)

[Writable reading buffers](#) (on page 6-36)

buffer.write.reading()

This function allows you to write readings into the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

For buffers that are set to the writable buffer style:

```
buffer.write.reading(bufferVar, readingValue)
buffer.write.reading(bufferVar, readingValue, seconds)
buffer.write.reading(bufferVar, readingValue, seconds, fractionalSeconds)
buffer.write.reading(bufferVar, readingValue, seconds, fractionalSeconds, status)
buffer.write.reading(bufferVar, readingValue, seconds, fractionalSeconds, status,
    "channel")
```

For buffers that are set to the full writable buffer style:

```
buffer.write.reading(bufferVar, readingValue, extraValue)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds,
    fractionalSeconds)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds,
    fractionalSeconds, status)
buffer.write.reading(bufferVar, readingValue, extraValue, seconds,
    fractionalSeconds, status, "channel")
```

<i>bufferVar</i>	The name of the buffer
<i>readingValue</i>	The first value that is recorded in the buffer index
<i>extraValue</i>	A second value that is recorded in the buffer index (only valid for buffer style WRITABLE_FULL)
<i>seconds</i>	An integer that represents the seconds
<i>fractionalSeconds</i>	The portion of the time that represents the fractional seconds
<i>status</i>	Additional information about the reading; see Details
<i>channel</i>	A string that specifies the channel to which to assign the data

Details

This command writes the data you specify into a reading buffer. The reading buffer must be set to the writable or full writable style, which is set when you make the buffer.

Data must be added in chronological order. If the time is not specified for a reading, it is set to one integer second after the last reading. As you write the data, the front-panel home screen updates and displays the reading you entered.

The *status* parameter provides additional information about the reading. The options are shown in the following table.

Parameter	Description
buffer.STAT_LIMIT1_HIGH	Reading is above high limit 1
buffer.STAT_LIMIT1_LOW	Reading is below low limit 1
buffer.STAT_LIMIT2_HIGH	Reading is above high limit 2
buffer.STAT_LIMIT2_LOW	Reading is below low limit 2

<code>buffer.STAT_ORIGIN</code>	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize)
<code>buffer.STAT_QUESTIONABLE</code>	Measure status questionable
<code>buffer.STAT_REL</code>	Relative offset
<code>buffer.STAT_SCAN</code>	Scan
<code>buffer.STAT_START_GROUP</code>	First reading in a group
<code>buffer.STAT_TERMINAL</code>	Measure terminal, front is 1, rear is 0

Example 1

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1)
buffer.write.reading(extBuffer, 2)
buffer.write.reading(extBuffer, 3)
buffer.write.reading(extBuffer, 4)
buffer.write.reading(extBuffer, 5)
buffer.write.reading(extBuffer, 6)
printbuffer(1, 6, extBuffer.readings, extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is writable.

Set the data format to show units of watts with 3½ digit resolution.

Write 6 pieces of data into the buffer.

Print the buffer, including the readings and units.

Read the buffer.

Output:

1, Watt DC, 2, Watt DC, 3, Watt DC, 4, Watt DC, 5, Watt DC, 6, Watt DC

Example 2

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
    buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
    extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is full writable.

Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.

Write 12 pieces of data into the buffer.

Print the buffer, including the readings and units.

Read the buffer.

Output:

```
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt
DC, 10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC
```

Example 3

```
reset()
eventlog.clear()
buf = buffer.make(10, buffer.STYLE_WRITABLE_FULL)
slot[1].pseudocard = 7700
slot[2].pseudocard = 7700
buffer.write.reading(buf, 10, 20, 1, 0.01, 0, "101")
buffer.write.reading(buf, 11, 21, 1, 0.1, 0, "120")
buffer.write.reading(buf, 11, 21, 2, 0.1, 0, "220")
printbuffer(1, 2, buf.channels)
```

Reset the instrument.

Clear the event log.

Create 10-point reading buffer named `buf` with a style of writable.

Set both slots to use a 7700 pseudocard.

Write data into the buffer.

Read the buffer.

Output:

```
101, 120
```

Also see

[buffer.make\(\)](#) (on page 14-22)

[buffer.write.format\(\)](#) (on page 14-34)

[Reading buffers](#) (on page 6-1)

[Writable reading buffers](#) (on page 6-36)

bufferVar.capacity

This attribute sets the number of readings a buffer can store.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
bufferCapacity = bufferVar.capacity
bufferVar.capacity = bufferCapacity
```

<i>bufferCapacity</i>	The maximum number of readings the buffer can store; set to 0 to maximize the buffer size (see Details)
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer

Details

This command allows you to change or view how many readings a buffer can store. Changing the size of a buffer will cause any existing data in the buffer to be lost.

If you select 0, the instrument creates the largest reading buffer possible based on the available memory when the buffer is created.

The overall capacity of all buffers stored in the instrument can be up to 6,000,000 readings for standard reading buffers.

For more information about buffer capacity, see [Setting reading buffer capacity](#) (on page 6-11).

Example

```
reset()
testData = buffer.make(500)
capTest = buffer.make(300)
bufferCapacity = capTest.capacity

print(bufferCapacity)

print(testData.capacity)

testData.capacity = 600
print(testData.capacity)
print(defbuffer1.capacity)
```

Create two user-defined reading buffers: *testData* and *capTest*.

Create a variable called *bufferCapacity* to hold the capacity of the *capTest* buffer.

Print *bufferCapacity*.

Output:

300

Print the capacity of *testData*.

Output:

500

Changes the capacity of *testData* to 600.

Print the capacity of *testData*.

Output:

600

Print the capacity of the default buffer *defbuffer1*.

Output:

100000

Also see

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.channels

This attribute contains the channels that produced the readings that are stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
date = bufferVar.channels[N]
```

<i>date</i>	The date of readings stored in <i>bufferVar</i> element <i>N</i>
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Example

```
reset()
-- Make a buffer named testData.
testData = buffer.make(50)

-- Set channels 1 to 9 to measure voltage with an NPLC of 0.1 and range of 10 V.
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE,
    dmm.ATTR_MEAS_NPLC, 0.1, dmm.ATTR_MEAS_RANGE, 10)

scan.create("101:109")
scan.scancount = 10

-- Set each scan to an interval of 1 second.
scan.scaninterval = 1.0

-- Start the scan
trigger.model.initiate()
waitcomplete()

-- Get the data.
dmm.measure.read(testData)
print(testData.channels[1])
```

Example output:
109

Also see

[buffer.delete\(\)](#) (on page 14-17)
[buffer.make\(\)](#) (on page 14-22)
[bufferVar.clear\(\)](#) (on page 14-43)
[print\(\)](#) (on page 14-368)
[printbuffer\(\)](#) (on page 14-369)
[Reading buffers](#) (on page 6-1)
[Remote buffer operation](#) (on page 6-30)

bufferVar.clear()

This function clears all readings and statistics from the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`bufferVar.clear()`

<code>bufferVar</code>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData)
testData.clear()
print("Readings in buffer after clear = "
      .. testData.n)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData)
```

Create a reading buffer named `testData`, make three readings and store them in `testData`, and then view the readings.

Print number of readings in `testData`.

Output:

-4.5010112303956e-10, -3.9923108222095e-12, -4.5013931471161e-10

Clear the readings in `testData`.

Verify that there are no readings in `testData`.

Output:

Readings in buffer after clear = 0

Store three new readings in `testData` and view those when complete.

Output:

4.923509754e-07, 3.332266330e-07, 3.974883867e-07

Also see

- [buffer.delete\(\)](#) (on page 14-17)
- [buffer.make\(\)](#) (on page 14-22)
- [bufferVar.clear\(\)](#) (on page 14-43)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

bufferVar.dates

This attribute contains the dates of readings that are stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
date = bufferVar.dates[N]
```

<i>date</i>	The date of readings stored in <i>bufferVar</i> element <i>N</i>
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<i>defbuffer1</i> or <i>defbuffer2</i>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

The dates are formatted as month, day, year.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 1, testData)
trigger.model.initiate()
waitcomplete()
print(testData.dates[1])
printbuffer(1, testData.n, testData.dates)
```

Create a reading buffer named *testData*, configure the instrument to make three measurements, and store the readings in the buffer.

Print the first reading date.

Example output:

```
11/27/2017
```

Prints the dates for readings 1 through the last reading in the buffer.

Example output:

```
11/27/2017, 11/27/2017,
11/27/2017
```

Also see

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.endindex

This attribute indicates the last index in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
endIndex = bufferVar.endindex
```

<i>endIndex</i>	Ending index of the buffer
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer

Details

Use this attribute to find the ending index in a reading buffer.

Example

```
test1 = buffer.make(100)
dmm.measure.count = 6
dmm.measure.read(test1)
print(test1.startindex, test1.endindex, test1.capacity)
dmm.measure.read(test1)
print(test1.startindex, test1.endindex)
```

Create a buffer named `test1` with a capacity of 100 readings.

Set the measure count to 6.

Make measurements and store them in buffer `test1`.

Get the start index, end index, and capacity of `test1`.

Output:

1, 6, 100

Make six more measurements and store them in buffer `test1`.

Get the start index and end index of `test1`.

Output:

1, 12

Also see

[bufferVar.startindex](#) (on page 14-60)

[buffer.make\(\)](#) (on page 14-22)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.extraformattedvalues

This attribute contains the measurement and the unit of measure of the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
extraFormat = bufferVar.extraformattedvalues[N]
```

<i>extraFormat</i>	The measurement and unit of measure of the extra values for readings
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<i>defbuffer1</i> or <i>defbuffer2</i>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the measurement and the unit of measure of an additional value, such as the sense voltage from a DC voltage ratio measurement. The reading buffer style must be set to full to use this option.

Example 1

```
reset()
mathExp = buffer.make(400, buffer.STYLE_FULL)
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
buffer.math(mathExp, buffer.UNIT_NONE, buffer.EXPR_MULTIPLY)
for x = 1, 3 do
    print("Reading: ", dmm.measure.read(mathExp))
end
display.changescreen(display.SCREEN_READING_TABLE)
print("Extra value reading 1: ", mathExp.extraformattedvalues[1])
print("Extra value reading 2: ", mathExp.extraformattedvalues[2])
print("Extra value reading 3: ", mathExp.extraformattedvalues[3])
```

Reset the instrument.

Make a buffer named *mathExp* set to hold 400 readings with a buffer style of FULL.

Set the measure function to DC voltage.

Set the buffer math expression to multiply readings against the previous readings.

Make three readings.

Display the reading table on the front panel of the instrument, where you can view the extra readings.

Print the extra values (the calculated values).

Example output:

```
Reading:      7.1233589551e-06
Reading:      7.1233080234e-06
Reading:      7.2616603575e-06
Extra value reading 1:  +7.1233590 u
Extra value reading 2:  +50.741880 p
Extra value reading 3:  +51.727043 p
```

Example 2

```
reset()  
testData = buffer.make(50, buffer.STYLE_FULL)  
dmm.measure.func = dmm.FUNC_DCV_RATIO  
dmm.measure.read(testData)  
printbuffer(1, testData.n, testData.extraformattedvalues)
```

Reset the instrument.

Create a reading buffer named `testData` that can hold a maximum of 50 readings and is set to the style full.

Make a measurement and save it to the `testData` buffer.

Print the first extra value with the unit of measure.

Example output:

-5.716896 RA

Also see

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.extravalues

This attribute contains the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
extraValue = bufferVar.extravalues[N]
```

<i>extraValue</i>	The extra values for readings
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains an additional value, such as the sense voltage from a DC voltage ratio measurement. The reading buffer style must be set to full to use this option.

Example 1

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
    buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.units, extBuffer.extravalues,
    extBuffer.units)
```

Creates a 100-point reading buffer named `extBuffer`. Style is full writable.
Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.
Write 12 pieces of data into the buffer.
Print the buffer, including the readings and units.
Read the buffer.
Output:
1, Watt DC, 7, Watt DC, 2, Watt DC, 8, Watt DC, 3, Watt DC, 9, Watt DC, 4, Watt DC, 10, Watt DC, 5, Watt DC, 11, Watt DC, 6, Watt DC, 12, Watt DC

Example 2

```
reset()
testData = buffer.make(50, buffer.STYLE_FULL)
dmm.measure.func = dmm.FUNC_DCV_RATIO
dmm.measure.read(testData)
print(testData.extravalues[1])
printbuffer(1, 1, testData.extravalues)
```

Reset the instrument.
Create a reading buffer named `testData` that can hold a maximum of 50 readings and is set to the style full.
Make a measurement and save it to the `testData` buffer.
Print the first extra reading value.
Example output:
-7.4235309424
-7.4235309424

Also see

- [buffer.delete\(\)](#) (on page 14-17)
- [buffer.make\(\)](#) (on page 14-22)
- [bufferVar.clear\(\)](#) (on page 14-43)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

bufferVar.extravalueunits

This attribute contains the units of the additional values in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
extraUnits = bufferVar.extravalueunits[N]
```

<i>extraUnits</i>	The units of the extra values for readings
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

This attribute contains the unit of measure of an additional value, such as the sense voltage from a DC voltage ratio measurement. The reading buffer style must be set to full to use this option.

Example 1

```
extBuffer = buffer.make(100, buffer.STYLE_WRITABLE_FULL)
buffer.write.format(extBuffer, buffer.UNIT_WATT, buffer.DIGITS_3_5,
    buffer.UNIT_WATT, buffer.DIGITS_3_5)
buffer.write.reading(extBuffer, 1, 7)
buffer.write.reading(extBuffer, 2, 8)
buffer.write.reading(extBuffer, 3, 9)
buffer.write.reading(extBuffer, 4, 10)
buffer.write.reading(extBuffer, 5, 11)
buffer.write.reading(extBuffer, 6, 12)
printbuffer(1, 6, extBuffer.readings, extBuffer.extravalueunits)
```

Creates a 100-point reading buffer named `extBuffer`. Style is full writable.

Set the data format to show units of watts with 3½ digit resolution for the first value and for the second value in the buffer index.

Write 12 pieces of data into the buffer.

Print the buffer, including the readings and extra value units.

Read the buffer.

Output:

1, Watt DC, 2, Watt DC, 3, Watt DC, 4, Watt DC, 5, Watt DC, 6, Watt DC

Example 2

```
reset()
testData = buffer.make(50, buffer.STYLE_FULL)
dmm.measure.func = dmm.FUNC_DCV_RATIO
dmm.measure.read(testData)
printbuffer(1, testData.n, testData.extravalueunits)
```

Reset the instrument.

Create a reading buffer named `testData` that can hold a maximum of 50 readings and is set to the style full.

Make a measurement and save it to the `testData` buffer.

Print the unit of measure of the first extra value.

Example output:

Ratio

Also see

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.fillmode

This attribute determines if a reading buffer is filled continuously or is filled once and stops.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	User-defined buffer: buffer.FILL_ONCE (0) defbuffer1: buffer.FILL_CONTINUOUS (1) defbuffer2: buffer.FILL_CONTINUOUS (1)

Usage

```
fillMode = bufferVar.fillmode
bufferVar.fillmode = fillMode
```

<i>fillMode</i>	Fill the buffer, then stop: <code>buffer.FILL_ONCE</code> or 0 Fill the buffer continuously: <code>buffer.FILL_CONTINUOUS</code> or 1
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer

Details

When a reading buffer is set to fill once, no data is overwritten in the buffer. When the buffer is filled, no more data is stored in that buffer and new readings are discarded.

When a reading buffer is set to fill continuously, the oldest data is overwritten by the newest data after the buffer fills.

When you change the fill mode of a buffer, any data in the buffer is cleared.

Example

```
reset()
testData = buffer.make(50)
print(testData.fillmode)
testData.fillmode = buffer.FILL_CONTINUOUS
print(testData.fillmode)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer. Print the fill mode setting for the `testData` buffer.

Output:
0

Set fill mode to continuous.

Print the fill mode setting for the `testData` buffer.

Output:
1

Also see

- [buffer.delete\(\)](#) (on page 14-17)
- [buffer.make\(\)](#) (on page 14-22)
- [bufferVar.clear\(\)](#) (on page 14-43)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

bufferVar.formattedreadings

This attribute contains the stored readings shown as numbers with units and prefixes.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
reading = bufferVar.formattedreadings[N]
```

<i>reading</i>	Buffer reading formatted with numbers and prefixes with units for element <i>N</i>
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

This read-only attribute is an array that contains the stored readings. The readings are shown as numbers with prefixes before the units symbol.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.formattedreadings[1])
printbuffer(1, testData.n, testData.formattedreadings)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.

Print the first reading.

Example output:

-0.0001901 V

Print all readings in the reading buffer.

Example output:

-0.0001901 V, +000.08537 mV, -000.13050 mV

Also see

- [bufferVar.readings](#) (on page 14-57)
- [buffer.delete\(\)](#) (on page 14-17)
- [buffer.make\(\)](#) (on page 14-22)
- [bufferVar.clear\(\)](#) (on page 14-43)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

bufferVar.fractionalseconds

This attribute contains the fractional second portion of the timestamp of each reading in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
fractionalSec = bufferVar.fractionalseconds[N]
```

<i>fractionalSec</i>	The fractional second portion of the timestamp to 1 ns resolution
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

This read-only attribute is an array of the fractional portion of the timestamp, in seconds, when each reading occurred. Seconds are shown as fractions.

Example

```

reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.fractionalseconds[1])
printbuffer(1, 6, testData.fractionalseconds)

```

Create a reading buffer named `testData` and make six measurements.

Print the fractional portion of the timestamp for the first reading in the buffer.

Example output:

0.647118937

Print the fractional portion of the timestamp for the first six readings in the buffer.

Example output:

0.647118937, 0.064543, 0.48196127, 0.89938724, 0.316800064, 0.734218263

Also see

[bufferVar.seconds](#) (on page 14-59)

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.logstate

This attribute indicates if information events are logged when the specified reading buffer is at 0% or 100% filled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Power cycle	Not applicable	defbuffer1: buffer.ON (1) defbuffer2: buffer.ON (1) User-created buffer: buffer.OFF (0)

Usage

```

logState = bufferVar.logstate
bufferVar.logstate = logState

```

<i>logState</i>	Do not log information events: <code>buffer.OFF</code> or 0 Log information events: <code>buffer.ON</code> or 1
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer

Details

If this is set to on, when the reading buffer is cleared (0% filled) or full (100% filled), an event is logged in the event log. If this is set to off, reading buffer status is not reported in the event log.

Example

```
reset()
MyBuffer = buffer.make(500)
print(MyBuffer.logstate)
```

Create the user-defined buffer `MyBuffer`.
Print the log state of `MyBuffer`.
Output:
0

Also see

[Using the event log](#) (on page 3-80)

bufferVar.n

This attribute contains the number of readings in the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
numberOfReadings = bufferVar.n
```

<pre>numberOfReadings</pre>	The number of readings stored in the reading buffer
<pre>bufferVar</pre>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer

Details

You can call this command to return the number of readings stored in the specified reading buffer.

You can use the `bufferVar.n` attribute in other commands. For example, to print all the readings in a buffer, use the following command:

```
printbuffer(1, bufferVar.n, bufferVar.readings)
```

Where `bufferVar` is the name of the buffer to use.

Example

```
reset()
testData = buffer.make(100)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.n)
print(defbuffer1.n)
print(defbuffer2.n)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.

Print the number of readings in `testData`.

Output:

3

Print the number of readings in `defbuffer1`.

Example output:

0

Print the number of readings in `defbuffer2`.

Example output:

0

Also see

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.readings

This attribute contains the readings stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
reading = bufferVar.readings[N]
```

<i>reading</i>	The value of the reading in the specified reading buffer
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 3, testData.readings)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.

Print the three readings in `testData`.

Output:

```
-9.6420389034124e-12, -4.5509945811872e-10, -9.1078204006445e-12
```

Also see

[bufferVar.n](#) (on page 14-55)

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.relativetimestamps

This attribute contains the timestamps, in seconds, when each reading occurred, relative to the timestamp of the first entry in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

`timestamp = bufferVar.relativetimestamps[N]`

<code>timestamp</code>	The timestamp, in seconds
<code>bufferVar</code>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<code>N</code>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Details

This read-only attribute is an array of timestamps, in seconds, of when each reading occurred relative to the timestamp of the first entry in the reading buffer. These timestamps are equal to the time that has lapsed for each reading since the first reading was stored in the buffer. Therefore, the relative timestamp for the first entry number in the reading buffer equals 0.

Example

<pre>reset() testData = buffer.make(50) trigger.model.load("SimpleLoop", 3, 0, testData) trigger.model.initiate() waitcomplete() print(testData.relativetimestamps[1]) printbuffer(1, 3, testData.relativetimestamps)</pre>	Create a reading buffer named <code>testData</code>, configure the instrument to make three measurements, and store the readings in the buffer. Print the relative timestamp for the first reading in the buffer. Example output: 0 Print the relative timestamp for the reading 1 through 3 in the buffer. Example output: 0, 0.383541, 0.772005

Also see

- [buffer.delete\(\)](#) (on page 14-17)
- [buffer.make\(\)](#) (on page 14-22)
- [bufferVar.clear\(\)](#) (on page 14-43)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

bufferVar.seconds

This attribute contains the timestamp of a reading in seconds, in UTC format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
nonFracSeconds = bufferVar.seconds[N]
```

<i>nonFracSeconds</i>	The nonfractional seconds portion of the timestamp when the reading was stored
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<i>defbuffer1</i> or <i>defbuffer2</i>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the nonfractional seconds portion of the timestamp when the reading was stored in Coordinated Universal Time (UTC) format.

The nonfractional seconds portion of the timestamp gives the lowest resolution down to 1 second. To access additional resolution of a timestamp, see *bufferVar.fractionalseconds*.

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 6, 0,
    testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 6, testData.seconds)
```

Create a reading buffer named *testData*, configure the instrument to make six measurements, and store the readings in the buffer.

Print the seconds portion for readings 1 to 6 in *testData*.

Example output:

```
1362261492, 1362261492,
1362261493, 1362261493,
1362261493, 1362261494
```

Also see

[bufferVar.fractionalseconds](#) (on page 14-53)

[bufferVar.relativetimestamps](#) (on page 14-58)

bufferVar.startindex

This attribute indicates the starting index in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
bufferVar.startindex = startIndex
```

<i>startIndex</i>	Starting index of the buffer
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<i>defbuffer1</i> or <i>defbuffer2</i>) or a user-defined buffer

Details

Use this attribute to find the starting index in a reading buffer.

Example

```
test1 = buffer.make(100)
dmm.measure.count = 6
dmm.measure.read(test1)
print(test1.startindex, test1.endindex, test1.capacity)
```

Create a buffer named `test1` with a capacity of 100 readings.
Set the measure count to 6.
Make measurements and store them in buffer `test1`.
Get the start index, end index, and capacity of `test1`.
Output:
`1, 6, 100`

Also see

- [bufferVar.endindex](#) (on page 14-45)
- [buffer.make\(\)](#) (on page 14-22)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

bufferVar.statuses

This attribute contains the status values of readings in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
statusInformation = bufferVar.statuses[N]
```

<i>statusInformation</i>	The status value when reading <i>N</i> of the specified buffer was acquired; refer to Details
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This read-only attribute is an array of status values for the readings in the buffer. The status values are floating-point numbers that encode the status value. Refer to the following table for values.

Buffer status bits for sense measurements

Bit (hex)	Name	Decimal	Description	<i>statusInformation</i> parameter
0x0001	STAT_QUESTIONABLE	1	Measure status questionable	buffer.STAT_QUESTIONABLE
0x0006	STAT_ORIGIN	6	A/D converter from which reading originated; for the DAQ6510, this will always be 0 (main) or 2 (digitize))	buffer.STAT_ORIGIN
0x0008	STAT_TERMINAL	8	Measure terminal, front is 1, rear is 0	buffer.STAT_TERMINAL
0x0010	STAT_LIMIT2_LOW	16	Measure status limit 2 low	buffer.STAT_LIMIT2_LOW
0x0020	STAT_LIMIT2_HIGH	32	Measure status limit 2 high	buffer.STAT_LIMIT2_HIGH
0x0040	STAT_LIMIT1_LOW	64	Measure status limit 1 low	buffer.STAT_LIMIT1_LOW
0x0080	STAT_LIMIT1_HIGH	128	Measure status limit 1 high	buffer.STAT_LIMIT1_HIGH
0x0100	STAT_START_GROUP	256	First reading in a group	buffer.STAT_START_GROUP

Example

```

reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 2, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, 2, testData.statuses)

```

Create a reading buffer named `testData`, configure the instrument to make two measurements, and store the readings in the buffer.

Print the status for the readings in `testData`.

Output:

64, 64

Indicating that the status is `buffer.STAT_LIMIT1_LOW`.

Also see

[buffer.make\(\)](#) (on page 14-22)

[buffer.delete\(\)](#) (on page 14-17)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.times

This attribute contains the time when the instrument made the reading.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
readingTime = bufferVar.times[N]
```

<i>readingTime</i>	The time of the reading in hours, minutes, and seconds
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <code>bufferVar.n</code> command to determine the number of readings in the buffer

Example

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.times[1])
printbuffer(1, 3, testData.times)
```

This example creates a reading buffer named `testData` and makes three measurements.

The `print()` command outputs the time of the first reading.

Output:
23:09:43

The `printbuffer()` command outputs the time of readings 1 to 3 in the reading buffer.

Output:
23:09:43, 23:09:43, 23:09:43

Also see

- [buffer.delete\(\)](#) (on page 14-17)
- [buffer.make\(\)](#) (on page 14-22)
- [bufferVar.clear\(\)](#) (on page 14-43)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

bufferVar.timestamps

This attribute contains the timestamp when each reading saved in the specified reading buffer occurred.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

readingTimestamp = *bufferVar.timestamps*[*N*]

<i>readingTimestamp</i>	The complete timestamp (including date, time, and fractional seconds) of reading number <i>N</i> in the specified reading buffer when the reading was acquired
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the timestamps (date, hours, minutes, seconds, and fractional seconds) of readings stored in the reading buffer.

Example 1

```
reset()
testData = buffer.make(50)
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
print(testData.timestamps[1])
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.

Print the first reading date.

Output:

```
03/01/2018 14:46:07.714614838
```

Example 2

```
for x = 1, 3 do printbuffer(x, x, testData.timestamps) end
```

For the buffer created in Example 1, print the timestamps for the readings.

Output:

```
03/01/2018 14:46:07.714614838
```

```
03/01/2018 14:46:08.100468838
```

```
03/01/2018 14:46:08.487631838
```

Also see

[buffer.delete\(\)](#) (on page 14-17)

[buffer.make\(\)](#) (on page 14-22)

[bufferVar.clear\(\)](#) (on page 14-43)

[print\(\)](#) (on page 14-368)

[printbuffer\(\)](#) (on page 14-369)

[Reading buffers](#) (on page 6-1)

[Remote buffer operation](#) (on page 6-30)

bufferVar.units

This attribute contains the unit of measure that is stored with readings in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Restore configuration Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
readingUnit = bufferVar.units[N]
```

<i>readingUnit</i>	Amp AC: AC current measurement Amp DC: DC current measurement Celsius: Temperature measurement Decibel: Units are set to decibel dBm: Decibel-milliwatt measurement Fahrenheit: Temperature measurement Farad: Capacitance measurement Hertz: Frequency measurement Kelvin: Temperature measurement X: Math is set to mx+b for the measurements Ohm: Resistance measurement %: Math is set to percent for the measurements Ratio: DCV ratio measurement Reciprocal: Math is set to reciprocal for the measurements Second: Period measurement Volt AC: AC voltage measurement Volt DC: DC voltage measurement Watt DC: Power measurement
<i>bufferVar</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>N</i>	The reading number <i>N</i> ; can be any value from 1 to the number of readings in the buffer; use the <i>bufferVar.n</i> command to determine the number of readings in the buffer

Details

This attribute contains the unit of measure that is stored with readings in the reading buffer.

Example

```
reset()
testData = buffer.make(50)
testData.fillmode = buffer.FILL_CONTINUOUS
dmm.measure.func = dmm.FUNC_DC_CURRENT
trigger.model.load("SimpleLoop", 3, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.units)
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.units)
```

Create a reading buffer named `testData`, configure the instrument to make three measurements, and store the readings in the buffer.
Set the buffer to fill continuously.
Set the measure function to current.
Make three readings.
Print the units for the readings.
Output:
Amp DC, Amp DC, Amp DC
Set the measure function to voltage.
Make three readings.
Output:
Volt DC, Volt DC, Volt DC

Also see

- [buffer.delete\(\)](#) (on page 14-17)
- [buffer.make\(\)](#) (on page 14-22)
- [bufferVar.clear\(\)](#) (on page 14-43)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)
- [Reading buffers](#) (on page 6-1)
- [Remote buffer operation](#) (on page 6-30)

channel.close()

This function closes the channels and channel pairs that are specified by the channel list parameter.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.close("channelList")
```

<i>channelList</i>	A string that lists the channels and channel pairs to close

Details

The action of the close command depends on which, if any, function is set for the DMM.

If no function is set, the listed channels or channel pairs are closed. You can select multiple channels.

If the DMM for the channel is set to a function, the listed channels or channel pairs are closed. In addition, it opens channels or channel pairs that could affect the measurements. When a channel is set to a function, only one channel can be specified in the channel list.

When you close a channel or channel pair, the instrument:

- Closes the items in the list of channels.
- Opens any channels on any slots that interfere with the measurement.
- Incurs the settling time and any user-specified delay.

This command is not available for digital I/O, digital-to-analog converter (DAC analog output), and totalizer channels. If the digital I/O, DAC analog output, or totalizer channel is in a range of channels, the unavailable channel is ignored. If you try to close it individually, an error is generated.

Use the `channel.getclose()` command to return a list of closed measurement channels, including the paired channels for 4-wire measurements.

Example 1

```
channel.close("101")
channel.close("102")
print(channel.getstate("101:105"))
```

This example assumes a 7706 module is installed with no functions set. This example closes channels 101, then closes 102 without opening 101. The state of channels 1 to 5 is as follows, indicating that channels 1 and 2 are closed and the others are open:

```
[1]=channel.IND_CLOSED, [2]=channel.IND_CLOSED, [3]=0, [4]=0, [5]=0
```

Example 2

```
reset()
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
print(channel.getdmm("101:105", dmm.ATTR_MEAS_FUNCTION))
```

```
channel.close("101")
print(channel.getstate("101:105"))
```

```
channel.close("102")
print(channel.getstate("101:105"))
```

This example assumes a 7706 module is installed. This example sets channels 1 to 5 on slot 1 to measure DC voltage.

Verify the DMM function settings. The output is:

```
[1]=dmm.FUNC_DC_VOLTAGE, [2]=dmm.FUNC_DC_VOLTAGE, [3]=dmm.FUNC_DC_VOLTAGE,
[4]=dmm.FUNC_DC_VOLTAGE, [5]=dmm.FUNC_DC_VOLTAGE
```

Close channel 101, then get the states of channels 101 to 105. The return is:

```
[1]=channel.IND_CLOSED, [2]=0, [3]=0, [4]=0, [5]=0
```

Close 102. Channel 101 is automatically opened, and the return is:

```
[1]=0, [2]=channel.IND_CLOSED, [3]=0, [4]=0, [5]=0
```

Also see

[channel.getclose\(\)](#) (on page 14-70)

[channel.getstate\(\)](#) (on page 14-79)

[channel.open\(\)](#) (on page 14-83)

[channel.setdelay\(\)](#) (on page 14-87)

channel.connectrule

This attribute controls the connection rule for closing and opening channels in the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Save setup	channel.CONCURRENT

Usage

```
rule = channel.connectrule
channel.connectrule = rule
```

rule

- `channel.BREAK_BEFORE_MAKE`: Break before make (BBM) connections for relays in the instrument
- `channel.MAKE_BEFORE_BREAK`: Make before break (MBB) connections for relays in the instrument
- `channel.CONCURRENT`: Does not guarantee a connection rule. The instrument closes relays as efficiently as possible to improve speed performance without applying a rule

Details

The connection rule describes the order in which switch channels are opened and closed during channel operations. This applies to scanning operation and when using the `channel.close()` command when a channel is set to a function.

The connection rule affects the operating time of these commands. These commands do not allow the instrument to continue execution until the settle time of the relays has expired.

When the connection rule is set to break before make, the instrument ensures that all switch channels open before any switch channels close. 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 make before break, 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.

With no connection rule (set to `channel.CONCURRENT`), 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. 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.

NOTE

You cannot guarantee the sequence of open and closure operations when the channel connect rule is set to `CONCURRENT`. It is highly recommended that you implement cold switching when the channel connect rule is set to `CONCURRENT`.

In general, the settling time of single commands that open and close switch channels depends on several 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:

- Break before make: Open settle time + close settle time
- Make before break: Close settle time + open settle time
- `channel.CONCURRENT`: Maximum of open settle time or close settle time

CAUTION

Make before break (also known as hot switching) can dry-weld reed relays so that they will always be on. Hot switching is recommended only when external protection is provided.

Example

```
channel.connectrule = channel.BREAK_BEFORE_MAKE
Sets the connect rule in the instrument to channel.BREAK_BEFORE_MAKE
```

Also see

[channel.close\(\)](#) (on page 14-66)

channel.getclose()

This function queries for the closed channels indicated by the channel list parameter.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
closed = channel.getclose()
closed = channel.getclose("channelList")
```

<i>closed</i>	A string that lists the channels that are presently closed in the specified channel list
<i>channelList</i>	A string that lists the channels to be queried (including <code>allslots</code> , <code>slot1</code> , or <code>slot2</code> to get information on all channels in both slots or a specific slot); returns all slots if nothing is specified

Details

Use this command to return a list of closed measurement channels, including the paired channel for 4-wire measurements. It does not return nonmeasurement channels.

If more than one channel is closed, they are delimited in the string.

If none of the channels in the channel list is closed, `nil` is returned.

Example

```
channelList = "101:103"  
channel.close("101")  
print(channel.getclose(channelList))  
channel.close("103")  
print(channel.getclose(channelList))
```

For this example, assume there is a card or pseudocard in slot 1 with no previously closed channels. The output is:

```
[1]=101  
[1]=101, [2]=103
```

Also see

[channel.close\(\)](#) (on page 14-66)

[channel.getstate\(\)](#) (on page 14-79)

[channel.open\(\)](#) (on page 14-83)

channel.getcommonsides()

This function lists the commonsides setting for the specified slot.

NOTE

This command is only applicable for cards that support commonsides measurements, such as the Keithley Instruments 7701 module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
state = channel.getcommonside("slot")
```

<i>state</i>	The state of the slot: - Common side disabled: 0 - Common side enabled: 1 - No card installed or card does not support commonside measurements: nil
<i>slot</i>	A string that lists the slot number; can be <code>allslots</code> , <code>slot1</code> , or <code>slot2</code>

Example

```
print(channel.getcommonside("slot1"))
```

Check the settings of channels 1 to 10 on slot 1. If common side is enabled, return is similar to:
`[1]=1, [2]=1, [3]=1, [4]=1, [5]=1, [6]=1, [7]=1, [8]=1, [9]=1, [10]=1`

Also see

[channel.setcommonside\(\)](#) (on page 14-84)

[slot\[slot\].commonsideohms](#) (on page 14-412)

channel.getcount()

This function returns the number of times the relays have been closed for the specified channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
counts = channel.getcount("channelList")
```

<i>counts</i>	A comma-delimited string that lists the number of times the specified channels have closed
<i>channelList</i>	A string listing the items to query, which can include: - Channels <code>slotX</code>, where <i>X</i> is 1 or 2 <code>allslots</code>

Details

The DAQ6510 keeps an internal count of the number of times each relay has been closed. This count can help you determine when relays require replacement. Refer to the switching module documentation for the contact life specifications for the relays.

If channels are specified, the count values are returned in the order in which the channels are specified. If slots are specified, the response lists the channels starting from lowest to highest. Because each slot is processed completely before going to the next, all slot 1 channels are listed before slot 2 channels.

Relay closures are counted only when a relay cycles from open to closed state.

It is good practice to get the relay count at the end of a program. This saves the latest count to memory.

Example

```
counts = channel.getcount("201:205")
print(counts)
```

Gets the close counts for channels 1 to 5 on slot 2.

Example output for channels 201, 202, 203, 204, and 205:

```
[1]=201, [2]=202, [3]=203, [4]=204, [5]=205
```

Also see

[channel.setcountinterval](#) (on page 14-86)

channel.getcountinterval()

This function returns the interval that is presently set for storage of channel relay closure counts.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
update = channel.getcountinterval()
```

update

The count update interval:

- 1 minute: `channel.COUNT_INTERVAL_1M`
- 5 minutes: `channel.COUNT_INTERVAL_5M`
- 10 minutes: `channel.COUNT_INTERVAL_10M`
- 15 minutes: `channel.COUNT_INTERVAL_15M`
- 30 minutes: `channel.COUNT_INTERVAL_30M`
- 60 minutes: `channel.COUNT_INTERVAL_60M`
- 24 hours: `channel.COUNT_INTERVAL_24H`

Example

```
print(channel.getcountinterval())
```

If the interval was set to 24 hours, the return is:
`channel.COUNT_INTERVAL_24H`

Also see

[channel.setcountinterval](#) (on page 14-86)

channel.getdelay()

This function queries for the additional delay time for the specified channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
delayTimes = channel.getdelay("channelList")
```

<i>delayTimes</i>	A comma-delimited string consisting of the delay times (in seconds) for channels specified in <i>channelList</i>
<i>channelList</i>	A string listing the channels to query for their delay times; slot1, slot2, or allslots allowed

Details

The delay times are returned in a comma-delimited list in the same order that the channels are specified in the channel list parameter. A value of zero (0) indicates that no additional delay time is incurred before a close command completes.

NOTE

Pseudocards do not support user delays, so this value is always zero (0) if a pseudocard is used.

Example

```
channel.setdelay("slot1", 3.1)
DelayTimes = channel.getdelay("107,105,103")
print(DelayTimes)
```

Set a delay of 3.1 s for all channels in slot 1. Query channels 7, 5, and 3 on that slot.

Also see

[channel.setdelay\(\)](#) (on page 14-87)

channel.getdmm()

This function returns the setting for a channel DMM attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.getdmm("channelList", setting)
```

<i>channelList</i>	List of channels to get from the DMM
<i>setting</i>	The DMM function or the parameter values to return for the selected channels

Details

You can retrieve one attribute at a time.

For detail on the options for *setting*, see the examples and lists in the `channel.setdmm` command.

Example

```
print(channel.getdmm("101", dmm.ATTR_MEAS_AUTO_DELAY))
```

Retrieve the auto delay setting for channel 1 in slot 1.

Example return:

```
[1]=dmm.DELAY_ON
```

Also see

[channel.setdmm](#) (on page 14-88)

channel.getlabel()

This function returns the label associated with the specified channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
label = channel.getlabel("channelNumber")
```

<i>label</i>	A string that lists the comma-delimited label for the channel specified in <i>channelNumber</i>
<i>channelNumber</i>	A string that lists the channel to query for the associated label

Details

Returns `nil` if no label is set.

Example

```
channel.setlabel('101', 'First')
channel.setlabel('102', 'Second')
channel.setlabel('103', 'Third')
print(channel.getlabel('101'))
print(channel.getlabel('102'))
print(channel.getlabel('103'))
```

Set the labels to First, Second, and Third.
Return the new labels.
Output:
First
Second
Third

Also see

[channel.setlabel\(\)](#) (on page 14-99)

channel.getmatch()

This function gets the match value on a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
matchValue = channel.getmatch("channelList")
```

<i>matchValue</i>	The match values set for the channels in <i>channelList</i>
<i>channelList</i>	String specifying digital I/O or totalizer channels to query, using normal channel list syntax

Details

This command returns the match value that was set for the specified channels.

Example

```
print(channel.getmatch("slot1"))
```

Assuming a 7706, query the match values set for digital I/O channels and totalizer on slot 1.
Output (assuming defaults):
[1]=0, [2]=0, [3]=0

Also see

[channel.setmatch\(\)](#) (on page 14-100)

channel.getmatchtype()

This function returns the match type for digital I/O and totalizer channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
matchType = channel.getmatchtype("channelList")
```

<i>matchType</i>	The comma-delimited states for channels in <i>channelList</i>
<i>channelList</i>	String specifying the digital I/O or totalizer channels to query, using normal channel list syntax

Details

The channel match types are:

- channel.MATCH_EXACT
- channel.MATCH_ANY
- channel.MATCH_UNCHANGED
- channel.MATCH_NONE

Example

```
print(channel.getmatchtype("121, 122, 125"))
```

Assuming a 7706, query the match type for channels 21, 22, and 25 on slot 1, which includes digital I/O and totalizer channels.
Example output:
[1]=channel.MATCH_NONE, [2]=channel.MATCH_NONE, [3]=channel.MATCH_UNCHANGED

Also see

[channel.setmatch\(\)](#) (on page 14-100)
[channel.setmatchtype\(\)](#) (on page 14-100)

channel.getmode()

This function returns the present mode of digital and totalizer channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
mode = channel.getmode("channelList")
```

<i>mode</i>	Return string of a comma-delimited list of modes
<i>channelList</i>	String that specifies the channels to query, using normal channel list syntax

Details

For digital channels, the return is:

- 8 if the channel is set to `channel.MODE_INPUT`
- 9 if the channel is set to `channel.MODE_OUTPUT`

For totalizer channels, the return is:

- 1 if the channel is set to `channel.MODE_RISING_EDGE`
- 0 if the channel is set to `channel.MODE_FALLING_EDGE`
- 3 if the channel is set to `channel.MODE_RISING_EDGE_READ_RESET`
- 2 if the channel is set to `channel.MODE_FALLING_EDGE_READ_RESET`

For DAC channels, the return is 17.

This command is not available for other types of channels. If other channels are included in a range, they are ignored. If they are specified directly, an error is generated. If you are querying a slot and the slot does not contain any channels with these types, an error is generated.

Example

```
print(channel.getmode("slot1"))
```

Query the configuration of the channels on slot 1. Assuming a 7706, channels 21 and 22 are digital I/O, 23 and 24 are DAC, and channel 25 is a totalizer.

Output:

```
[1]=1, [2]=1, [3]=17, [4]=17, [5]=2
```

Also see

[channel.setmode\(\)](#) (on page 14-102)

channel.getstate()

This function returns the state indicators of the channels in the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
state = channel.getstate()  
state = channel.getstate("channelList")
```

state	Return a comma-delimited string that lists the states for the channels in <i>channelList</i> ; refer to Details
channelList	String specifying the channels to query; use normal channel list syntax; if no channels are defined, all slots are returned

Details

This command returns the overload, match, closed, or open state of a channel. The states that can be returned depend on the type of channel.

All channels can report an open or closed channel.

Totalizer and digital I/O channels can report that a value has been matched.

Totalizer channels can also report that the count has overflowed, which means the last value read was less than the previous value read. This occurs when the totalizer reaches 4,294,967,295 and automatically resets to zero between reads.

Cards are returned sequentially by channel number.

Each bit in the return represents a different indicator. Therefore, multiple indicators can be present (the OR operation is performed bitwise).

Possible returns are:

- 0: Channel is open
- channel.IND_CLOSED: Channel is closed
- channel.IND_MATCH: Digital I/O or totalizer channel value is matched
- channel.IND_OVERFLOW: Totalizer channel has overflowed

Example

```
channel.close("105")  
State = channel.getstate("101:120")  
print(State)
```

Close channel 5 on slot 1.

Query the state of the first 20 channels on slot 1.

View the response assigned to *State*.

Output (assuming a 7706):

```
[1]=0, [2]=0, [3]=0, [4]=0, [5]=channel.IND_CLOSED, [6]=0, [7]=0, [8]=0, [9]=0,  
[10]=0, [11]=0, [12]=0, [13]=0, [14]=0, [15]=0, [16]=0, [17]=0, [18]=0,  
[19]=0, [20]=0
```

Also see

[channel.getclose\(\)](#) (on page 14-70)

[channel.setmatch\(\)](#) (on page 14-100)

channel.gettype()

This function returns the type associated with a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
type = channel.gettype("channelList")
```

<i>type</i>	Returns a comma-delimited list of the type of channels in <i>channelList</i>
<i>channelList</i>	String specifying the channels to query, using normal <i>channelList</i> syntax

Details

The channel type is defined by the physical hardware of the card on which the channel exists. The following are valid channel types:

- `channel.TYPE_BACKPLANE`
- `channel.TYPE_DAC`
- `channel.TYPE_DIGITAL`
- `channel.TYPE_POLE`
- `channel.TYPE_RADIO`
- `channel.TYPE_SWITCH`
- `channel.TYPE_TOTALIZER`

Refer to the card-specific documentation for more information about the channel types available for your card.

Example

<pre>print(channel.gettype("101, 125"))</pre>
Query the channel type of channels 1 and 25 in slot 1, assuming a 7706.
Output:
<pre>[1]=channel.TYPE_SWITCH, [2]=channel.TYPE_TOTALIZER</pre>

Also see

None

channel.getwidth()

This function gets the width used by the channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
width = channel.getwidth("channelNumber")
```

<i>width</i>	The width used by the channel
<i>channelNumber</i>	The first channel number of the channels to be combined

Details

Changing the width allows for multiple sequential digital I/O channels to be linked. This option only applies to digital I/O channels, such as channels 21 and 22 on the 7706 and channels 11 to 14 on the 7707.

For widths greater than 1, the specified channel occupies the least significant byte. For example, writing the value of 4278255360 (hexadecimal FF00FF00) to channel 1 with a width of 4 sets channel 1 to 0, channel 2 to 255 (hexadecimal FF), channel 3 to 0, and channel 4 to 255 (hexadecimal FF). Writing the value of 4278255360 to channel 1 with a width of 1 sets channel 1 to 0 and leaves other channels untouched.

You must use decimal values when sending commands to the DAQ6510.

The maximum width is the number of digital I/O channels available on the card. If a width crosses channels, then only the channels set to output are affected.

Example

```
print(channel.getwidth("121"))
```

Returns the width setting for channel 21 of slot 1. If the width was set to 2, the return is:
[1]=2

Also see

[channel.setwidth\(\)](#) (on page 14-103)

channel.multiple.close()

This function closes the listed channels without affecting any other channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.multiple.close("channelList")
```

<i>channelList</i>	The list of channels to close

Details

This command closes the specified channels without affecting any other channels, including paired channels.

If the channel list is large, you should use the `opc()` function with the multiple close.

Example

```
channel.multiple.close("101:109")
```

Close channels 1 to 9 on slot 1. Paired channels and backplane relays are not affected.

Also see

[channel.getclose\(\)](#) (on page 14-70)

channel.multiple.open()

This function opens the channels in the channel list without affecting any others.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.multiple.open("channelList")
```

<i>channelList</i>	A list of the channels to open

Details

Opens only the specified channels. Backplane relays and paired channels are not affected.

Example

```
channel.multiple.open("102, 103, 104")
```

Open channels 102, 103, and 104.

Also see

[channel.multiple.close\(\)](#) (on page 14-82)

channel.open()

This command opens the specified channels and channel pairs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.open("channelList")
```

channelList

String listing the channels to open; `allslots`, `slot1`, and `slot2` available, which open all channels on all slots, all channels on slot 1, or all channels on slot 2

Details

If the specified channels are not set to a measurement function, this command opens the specified channels without affecting other channels.

If the specified channels are set to a measurement function, their paired channels and backplane channels are also opened.

The settling time associated with a channel must elapse before the command completes. User delay is not added when a relay opens.

You cannot open digital I/O, DAC, and totalizer channels. If you include one of these channels in the channel list, the invalid channel is ignored but other channels open normally.

Example 1

```
channel.close("101")
channel.close("102")
channel.open("allslots")
```

This example closes channels 101 and 102, then opens all channels.

Example 2

```
channel.open("slot1")
```

Opens all channels on slot 1.

Example 3

```
channel.open("allslots")
```

Opens all channels on all slots.

Also see

- [channel.close\(\)](#) (on page 14-66)
- [channel.getclose\(\)](#) (on page 14-70)
- [channel.getdelay\(\)](#) (on page 14-74)
- [channel.getstate\(\)](#) (on page 14-79)
- [channel.setdelay\(\)](#) (on page 14-87)

channel.read()

This function reads a value from a totalizer, DAC, or digital I/O channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = channel.read("channelList")
value = channel.read("channelList", readingBuffer)
```

value	Return string that lists the comma-delimited read values for the channels in <i>channelList</i>
channelList	String that specifies a list of channels, using channel list notation
readingBuffer	Reading buffer in which to store read values

Details

For totalizer channels, if the mode is set to a reset mode, the count is reset when this command is sent.

Example

<pre>print(channel.read("125"))</pre>	Assuming a 7706, read the count from the totalizer channel.

Also see

None

channel.setcommonside()

This function sets up the specified slot to have a common side path.

NOTE

This command is only applicable for cards that support commonside measurements, such as the Keithley Instruments 7701 module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`channel.setcommonsides("slot", state)`

<code>slot</code>	A string that lists the slot number; can be <code>allslots</code> , <code>slot1</code> , or <code>slot2</code>
<code>state</code>	<code>channel.ON</code> or <code>channel.OFF</code>

Details

Set commonsides ohms before setting the function.

Example

```
channel.open("allslots")
channel.setcommonsides("slot1", channel.ON)
channel.setdmm("slot1", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_4W_RESISTANCE)
channel.close("101")
print(dmm.measure.read())
channel.close("132")
print(dmm.measure.read())
printbuffer(1, defbuffer1.n, defbuffer1)
channel.open("allslots")
```

Open all slots.
Set the channels on slot 1 to have a common side path.
Set slot 1 to measure 4-wire resistance.
Close channel 101.
Make a measurement.
Repeat closing and reading channels for each DUT (channels 102 to 131).

Also see

- [channel.getcommonsides\(\)](#) (on page 14-71)
- [slot\[slot\].commonsidesohms](#) (on page 14-412)

channel.setcountinterval()

This function determines how often the instrument stores the channel relay closure count.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	None	Nonvolatile memory	channel.COUNT_INTERVAL_15M

Usage

```
channel.setcountinterval(interval)
```

interval

The count update interval:

- 1 minute: channel.COUNT_INTERVAL_1M
- 5 minutes: channel.COUNT_INTERVAL_5M
- 10 minutes: channel.COUNT_INTERVAL_10M
- 15 minutes: channel.COUNT_INTERVAL_15M
- 30 minutes: channel.COUNT_INTERVAL_30M
- 60 minutes: channel.COUNT_INTERVAL_60M
- 24 hours: channel.COUNT_INTERVAL_24H

Details

This command determines how often channel relay closure counts are stored in nonvolatile memory. If the DAQ6510 is turned off before the count is stored, the counts are lost.

The DAQ6510 stores relay closure counts:

- At the interval set by this command.
- When the counts are queried.

Relay closure counts are updated in temporary memory every time a channel is closed regardless of how it was closed.

The lower the interval, the less chance there is of losing relay counts due to power failures. However, writing to memory more often may reduce scan throughput.

It is good practice to query the count at the end of a program to manually update the count.

Example

```
channel.setcountinterval(channel.COUNT_INTERVAL_30M)
```

Sets the channel count save interval to 30 minutes.

Also see

[channel.getcount\(\)](#) (on page 14-72)

[channel.getcountinterval\(\)](#) (on page 14-73)

channel.setdelay()

This function sets additional delay time for specified channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Recall setup Power cycle	Create configuration script Save setup	0 (0 s)

Usage

```
channel.setdelay("channelList", delay)
```

<i>channelList</i>	A string listing the channels for which to add delay time
<i>delay</i>	Delay time for the selected channels; minimum is 0 seconds

Details

After a channel closes, a command incurs the delay time indicated in the response for a channel before it completes. However, the internal settling time must elapse before the user delay is incurred. Therefore, the sequence is:

1. Command is processed
2. Channel closes
3. Settling time is incurred
4. Channel delay is incurred
5. Command completes

The channel delay is an additional delay that is added after a channel is closed. You can use this delay to allow additional settling time for a signal on that channel. For most cards, the resolution of the delay is 10 μ s. However, check the documentation for your card to verify. To see if the delay value was modified after setting, query the value.

Setting a delay only applies to switch channels.

The delay being specified may be updated based on the delay resolution of the card.

To query the delay value, use the `channel.getdelay()` command. Pseudocards do not replicate the additional delay time.

Example

```
channel.setdelay("103, 105", 50e-6)
Sets channel 3 and 5 of slot 1 for a delay time of 50  $\mu$ s.
```

Also see

[channel.getdelay\(\)](#) (on page 14-74)

channel.setdmm()

This function configures the DMM for a channel or group of channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
channel.setdmm("channelList", setting, value)
channel.setdmm("channelList", setting, value, ..., setting, value)
```

<i>channelList</i>	List of channels for which to set the DMM
<i>setting</i>	The DMM function or the parameter for the function to be applied to the <i>channelList</i> ; refer to Details and the tables following the examples
<i>value</i>	The function or attribute value

Details

You must use this command to set the measure or digitize function before using it to set the parameters for the selected function. You can send up to four *setting* and *value* pairs for this command.

To set up a measure function, assign *setting* to `dmm.ATTR_MEAS_FUNCTION` and set the *value* to one of the options in the table below. For example, to set channel 1 in slot 1 to the DC voltage function, you send:

```
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
```

When a function is assigned to a channel, the default values of the new function are assigned to the channel.

Options for `dmm.ATTR_MEAS_FUNCTION`

<code>dmm.FUNC_DC_VOLTAGE</code>	<code>dmm.FUNC_RESISTANCE</code>	<code>dmm.FUNC_ACV_FREQUENCY</code>
<code>dmm.FUNC_AC_VOLTAGE</code>	<code>dmm.FUNC_4W_RESISTANCE</code>	<code>dmm.FUNC_ACV_PERIOD</code>
<code>dmm.FUNC_DC_CURRENT</code>	<code>dmm.FUNC_DIODE</code>	<code>dmm.FUNC_DCV_RATIO</code>
<code>dmm.FUNC_AC_CURRENT</code>	<code>dmm.FUNC_CAPACITANCE</code>	
<code>dmm.FUNC_TEMPERATURE</code>	<code>dmm.FUNC_CONTINUITY</code>	

To set up a digitize function, assign *setting* to `dmm.ATTR_DIGI_FUNCTION`. Set the *value* to either `dmm.FUNC_DIGITIZE_CURRENT` or `dmm.FUNC_DIGITIZE_VOLTAGE`. For example, to set channel 3 in slot 1 to the digitize voltage function, you send:

```
channel.setdmm("103", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
```

Once the function is set, you can set the parameters and settings for that function. The following lists describe the settings that are available for each function, with links to the descriptions of the corresponding TSP command descriptions. The options for each function setting are the same as the settings for the TSP commands.

Example 1

```
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101", dmm.ATTR_MEAS_NPLC, 1)
channel.setdmm("102", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_RESISTANCE)
channel.setdmm("102", dmm.ATTR_MEAS_NPLC, 1)
channel.close("101")
print(dmm.measure.read())
channel.close("102")
print(dmm.measure.read())
```

Set up channel 1 on slot 1 to measure DC voltage and channel 2 to measure 2-wire resistance. Close each channel and make a measurement.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 1)
channel.close("101")
print(dmm.measure.read())
channel.close("102")
print(dmm.measure.read())
```

Set channels 1 to 9 on slot 1 to measure DC voltage.

Set channels 1 to 9 to use an NPLC of 1.

Close channel 1 and make a measurement.

Close channel 2 and make a measurement.

Example 3

```
channel.setdmm("101:109",
    dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE,
    dmm.ATTR_MEAS_NPLC, 1,
    dmm.ATTR_MEAS_RANGE_AUTO, dmm.ON,
    dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_4_5)
```

This command sets channels 1 to 9 on slot 1 to measure DC voltage using an NPLC of 1, with autorange on, and at 4½ digit resolution.

Example 4

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 1)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE_AUTO, dmm.ON)
channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_4_5)
```

These commands set the same values as Example 3, except that each value is sent in a separate command.

Also see

[channel.getdmm](#) (on page 14-75)

DC voltage (dmm.FUNC_DC_VOLTAGE)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE
[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO
[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO
[dB reference](#) (on page 14-223): dmm.ATTR_MEAS_DB_REFERENCE
[dBm reference](#) (on page 14-225): dmm.ATTR_MEAS_DBM_REFERENCE
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Input impedance](#) (on page 14-243): dmm.ATTR_MEAS_INPUT_IMPEDANCE
[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC
[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE
[Unit](#) (on page 14-324): dmm.ATTR_MEAS_UNIT
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_N

AC voltage (dmm.FUNC_AC_VOLTAGE)

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO
[dB reference](#) (on page 14-223): dmm.ATTR_MEAS_DB_REFERENCE
[dBm reference](#) (on page 14-225): dmm.ATTR_MEAS_DBM_REFERENCE
[Detector bandwidth](#) (on page 14-227): dmm.ATTR_MEAS_DETECTBW
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE
[Unit](#) (on page 14-324): dmm.ATTR_MEAS_UNIT
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_N

DC current (dmm.FUNC_DC_CURRENT)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO

[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC

[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE

[User delay](#) (on page 14-326) N (where N is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ N

AC current (dmm.FUNC_AC_CURRENT)

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO

[Detector bandwidth](#) (on page 14-227): dmm.ATTR_MEAS_DETECTBW

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE

[User delay](#) (on page 14-326) N (where N is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ N

Temperature (dmm.FUNC_TEMPERATURE)

[2-wire RTD type](#) (on page 14-322): dmm.ATTR_MEAS_TWO_RTD

[3-wire RTD type](#) (on page 14-315): dmm.ATTR_MEAS_THREE_RTD

[4-wire RTD type](#) (on page 14-238): dmm.ATTR_MEAS_FOUR_RTD

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC

[Offset Compensation](#) (on page 14-271): dmm.ATTR_MEAS_OFFCOMP_ENABLE

[Open lead detector](#) (on page 14-273): dmm.ATTR_MEAS_OPEN_DETECTOR

[Reference junction](#) (on page 14-280): dmm.ATTR_MEAS_REF_JUNCTION

[RTD Alpha](#) (on page 14-289): dmm.ATTR_MEAS_RTD_ALPHA

[RTD Beta](#) (on page 14-291): dmm.ATTR_MEAS_RTD_BETA

[RTD Delta](#) (on page 14-293): dmm.ATTR_MEAS_RTD_DELTA

[RTD Zero](#) (on page 14-295): dmm.ATTR_MEAS_RTD_ZERO

[Simulated reference temperature](#) (on page 14-309): dmm.ATTR_MEAS_SIM_REF_TEMP

[Thermistor](#) (on page 14-311): dmm.ATTR_MEAS_THERMISTOR

[Thermocouple](#) (on page 14-313): dmm.ATTR_MEAS_THERMOCOUPLE

[Transducer](#) (on page 14-320): dmm.ATTR_MEAS_TRANSDUCER

[Unit](#) (on page 14-324): dmm.ATTR_MEAS_UNIT

[User delay](#) (on page 14-326) N (where N is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ N

2-wire resistance (dmm.FUNC_RESISTANCE)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE
[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO
[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC
[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

4-wire resistance (dmm.FUNC_4W_RESISTANCE)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE
[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO
[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC
[Offset compensation](#) (on page 14-271): dmm.ATTR_MEAS_OFFCOMP_ENABLE
[Open lead detector](#) (on page 14-273): dmm.ATTR_MEAS_OPEN_DETECTOR
[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Diode (dmm.FUNC_DIODE)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE
[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO
[Bias level](#) (on page 14-212): dmm.ATTR_MEAS_BIAS_LEVEL
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Capacitance (dmm.FUNC_CAPACITANCE)

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Continuity (dmm.FUNC_CONTINUITY)

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Frequency (dmm.FUNC_ACV_FREQUENCY)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE
[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[Threshold autorange](#) (on page 14-317): dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO
[Threshold range](#) (on page 14-319): dmm.ATTR_MEAS_THRESHOLD_RANGE
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Period (dmm.FUNC_ACV_PERIOD)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[Threshold autorange](#) (on page 14-317): dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO

[Threshold range](#) (on page 14-319): dmm.ATTR_MEAS_THRESHOLD_RANGE

[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ *N*

DCV ratio (dmm.FUNC_DCV_RATIO)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO

[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC

[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE

[Sense range \(read only\)](#) (on page 14-298): dmm.ATTR_MEAS_SENSE_RANGE

[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ *N*

Digitize current (dmm.FUNC_DIGITIZE_CURRENT)

[Aperture](#) (on page 14-148): dmm.ATTR_DIGI_APERTURE

[Count](#) (on page 14-150): dmm.ATTR_DIGI_COUNT

[Display digits](#) (on page 14-154): dmm.ATTR_DIGI_DIGITS

[Range](#) (on page 14-179): dmm.ATTR_DIGI_RANGE

[Sample rate](#) (on page 14-187): dmm.ATTR_DIGI_SAMPLE_RATE

[Unit](#) (on page 14-189): dmm.ATTR_DIGI_UNIT

[User delay](#) (on page 14-190) *N* (where *N* is 1 to 5): dmm.ATTR_DIGI_USER_DELAY_ *N*

Digitize voltage (dmm.FUNC_DIGITIZE_VOLTAGE)

[Aperture](#) (on page 14-148): dmm.ATTR_DIGI_APERTURE

[Count](#) (on page 14-150): dmm.ATTR_DIGI_COUNT

[Decibel reference](#) (on page 14-152): dmm.ATTR_DIGI_DB_REFERENCE

[Decibel-milliwatts reference](#) (on page 14-153): dmm.ATTR_DIGI_DBM_REFERENCE

[Display digits](#) (on page 14-154): dmm.ATTR_DIGI_DIGITS

[Input impedance](#) (on page 14-157): dmm.ATTR_DIGI_INPUT_IMPEDANCE

[Range](#) (on page 14-179): dmm.ATTR_DIGI_RANGE

[Relative enable](#) (on page 14-184): dmm.ATTR_DIGI_REL_ENABLE

[Relative level](#) (on page 14-185): dmm.ATTR_DIGI_REL_LEVEL

[Sample rate](#) (on page 14-187): dmm.ATTR_DIGI_SAMPLE_RATE

[Unit](#) (on page 14-189): dmm.ATTR_DIGI_UNIT

[User delay](#) (on page 14-190) *N* (where *N* is 1 to 5): dmm.ATTR_DIGI_USER_DELAY_1

Math options (measure)

[Enable math](#) (on page 14-258): dmm.ATTR_MEAS_MATH_ENABLE

[b \(offset\) value](#) (on page 14-263): dmm.ATTR_MEAS_MATH_MXB_BF

[m \(scalar\) value](#) (on page 14-265): dmm.ATTR_MEAS_MATH_MXB_MF

[Math format](#) (on page 14-261): dmm.ATTR_MEAS_MATH_FORMAT

[Percent](#) (on page 14-267): dmm.ATTR_MEAS_MATH_PERCENT

Math options (digitize)

[Enable math](#) (on page 14-169): dmm.ATTR_DIGI_MATH_ENABLE

[b \(offset\) value](#) (on page 14-174): dmm.ATTR_DIGI_MATH_MXB_BF

[m \(scalar\) value](#) (on page 14-176): dmm.ATTR_DIGI_MATH_MXB_MF

[Math format](#) (on page 14-172): dmm.ATTR_DIGI_MATH_FORMAT

[Percent](#) (on page 14-177): dmm.ATTR_DIGI_MATH_PERCENT

Limit options (measure)

[Limit 1 audible](#) (on page 14-245): `dmm.ATTR_MEAS_LIMIT_AUDIBLE_1`

[Limit 1 auto clear](#) (on page 14-246): `dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_1`

[Limit 1 enable](#) (on page 14-248): `dmm.ATTR_MEAS_LIMIT_ENABLE_1`

[Limit 1 fail](#) (on page 14-250): `dmm.ATTR_MEAS_LIMIT_FAIL_1`

[Limit 1 high value](#) (on page 14-251): `dmm.ATTR_MEAS_LIMIT_HIGH_1`

[Limit 1 low value](#) (on page 14-252): `dmm.ATTR_MEAS_LIMIT_LOW_1`

[Limit 2 audible](#) (on page 14-245): `dmm.ATTR_MEAS_LIMIT_AUDIBLE_2`

[Limit 2 auto clear](#) (on page 14-246): `dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_2`

[Limit 2 enable](#) (on page 14-248): `dmm.ATTR_MEAS_LIMIT_ENABLE_2`

[Limit 2 fail](#) (on page 14-250): `dmm.ATTR_MEAS_LIMIT_FAIL_2`

[Limit 2 high value](#) (on page 14-251): `dmm.ATTR_MEAS_LIMIT_HIGH_2`

[Limit 2 low value](#) (on page 14-252): `dmm.ATTR_MEAS_LIMIT_LOW_2`

Limit options (digitize)

[Limit 1 audible](#) (on page 14-158): `dmm.ATTR_DIGI_LIMIT_AUDIBLE_1`

[Limit 1 auto clear](#) (on page 14-159): `dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_1`

[Limit 1 enable](#) (on page 14-162): `dmm.ATTR_DIGI_LIMIT_ENABLE_1`

[Limit 1 fail](#) (on page 14-163): `dmm.ATTR_DIGI_LIMIT_FAIL_1`

[Limit 1 high value](#) (on page 14-164): `dmm.ATTR_DIGI_LIMIT_HIGH_1`

[Limit 1 low value](#) (on page 14-165): `dmm.ATTR_DIGI_LIMIT_LOW_1`

[Limit 2 audible](#) (on page 14-158): `dmm.ATTR_DIGI_LIMIT_AUDIBLE_2`

[Limit 2 auto clear](#) (on page 14-159): `dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_2`

[Limit 2 enable](#) (on page 14-162): `dmm.ATTR_DIGI_LIMIT_ENABLE_2`

[Limit 2 fail](#) (on page 14-163): `dmm.ATTR_DIGI_LIMIT_FAIL_2`

[Limit 2 high value](#) (on page 14-164): `dmm.ATTR_DIGI_LIMIT_HIGH_2`

[Limit 2 low value](#) (on page 14-165): `dmm.ATTR_DIGI_LIMIT_LOW_2`

Analog trigger settings (measurement functions)

[Edge level](#) (on page 14-191): `dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL`

[Edge slope](#) (on page 14-193): `dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE`

[Mode](#) (on page 14-140): `dmm.ATTR_DIGI_ATRIG_MODE`

[Window direction](#) (on page 14-197): `dmm.ATTR_MEAS_ATRIG_WINDOW_DIRECTION`

[Window level high](#) (on page 14-199): `dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_HIGH`

[Window level low](#) (on page 14-201): `dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_LOW`

Analog trigger settings (digitize functions)

[Edge level](#) (on page 14-137): `dmm.ATTR_DIGI_ATRIG_EDGE_LEVEL`

[Edge slope](#) (on page 14-138): `dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE`

[Mode](#) (on page 14-140): `dmm.ATTR_DIGI_ATRIG_MODE`

[Window direction](#) (on page 14-142): `dmm.ATTR_DIGI_ATRIG_WINDOW_DIRECTION`

[Window level high](#) (on page 14-144): `dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_HIGH`

[Window level low](#) (on page 14-146): `dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_LOW`

Filter options (measure only)

[Filter enable](#) (on page 14-232): `dmm.ATTR_MEAS_FILTER_ENABLE`

[Filter count](#) (on page 14-230): `dmm.ATTR_MEAS_FILTER_COUNT`

[Filter type](#) (on page 14-234): `dmm.ATTR_MEAS_FILTER_TYPE`

[Filter window](#) (on page 14-236): `dmm.ATTR_MEAS_FILTER_WINDOW`

Relative offset settings (measurement functions)

[Relative offset enable](#) (on page 14-283): `dmm.ATTR_MEAS_REL_ENABLE`

[Relative offset method](#) (on page 14-287) (DCV ratio measurements only):
`dmm.ATTR_MEAS_REL_METHOD`

[Relative offset value](#) (on page 14-284): `dmm.ATTR_MEAS_REL_LEVEL`

Relative offset settings (digitize functions)

[Relative offset enable](#) (on page 14-184): `dmm.ATTR_DIGI_REL_ENABLE`

[Relative offset value](#) (on page 14-185): `dmm.ATTR_DIGI_REL_LEVEL`

NOTE

When you are using relative offset on a closed channel, make sure to set the relative offset level on that channel before enabling the offset.

Terminal options

[Terminal](#) (on page 14-327) for measurement functions: `dmm.ATTR_MEAS_TERMINALS`

[Terminal](#) (on page 14-327) for digitize functions: `dmm.ATTR_DIGI_TERMINALS`

channel.setlabel()

This function sets the label associated with a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Recall setup Power cycle	Create configuration script Save setup	No label

Usage

```
channel.setlabel("channelNumber", "labelname")
```

<i>channelNumber</i>	A string that lists the channel for which to set the label
<i>labelname</i>	A string that contains the label for the channel in <i>channelNumber</i> , up to 19 characters

Details

This command sets the label of the specified channel to the label value. The label must be unique; you cannot assign the same label to more than one channel. Labels cannot start with a digit. They can be up to 19 characters. On the front panel of the instrument, only the first few characters are displayed.

To clear a label, set it to an empty string ("").

After defining a label, you can use it to specify the channel instead of using the channel number in commands.

Example

<pre>channel.setlabel('101', 'First') channel.setlabel('102', 'Second') channel.setlabel('103', 'Third') print(channel.getlabel('101')) print(channel.getlabel('102')) print(channel.getlabel('103')) print(channel.getlabel('First'))</pre>	Set the labels to First, Second, and Third. Return the new labels. Output: First Second Third First

Also see

[channel.getlabel\(\)](#) (on page 14-76)

channel.setmatch()

This function sets the match value on a digital input or totalizer channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore setup Instrument reset Power cycle	Save setup	0

Usage

```
channel.setmatch("channelList", matchValue)
channel.setmatch("channelList", matchValue, mask)
```

<i>channelList</i>	String that specifies the channels to query, using normal channel list syntax
<i>matchValue</i>	Channel value to compare on the specified channel
<i>mask</i>	Value to specify the bits used to mask <i>matchValue</i>

Details

You can use matching to set a state or generate an event when the digital I/O or totalizer count matches a value, instead of continually reading the digital input or totalizer.

The default mask is all bits.

Switch channels are not supported. If they are included in a range or slot specifier, they are ignored. If they are specified directly, an error is generated.

Example

```
channel.setmatchtype("121", channel.MATCH_EXACT)
channel.setmatch("121", 32)
```

Assuming a 7706 is installed, set the match type for digital I/O channel 21 on slot 1 to exact. Set the match value to 32.

Also see

[channel.getmatch\(\)](#) (on page 14-76)

channel.setmatchtype()

This function sets the match type on a digital I/O or totalizer channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	channel.MATCH_NONE

Usage

```
channel.setmatchtype("channelList", type)
```

<i>channelList</i>	String specifying the channels to set, using normal channel list syntax
<i>type</i>	A value for setting the match operation used on this specific channel

Details

The types of match values include:

- `channel.MATCH_EXACT`
- `channel.MATCH_ANY`
- `channel.MATCH_UNCHANGED`
- `channel.MATCH_NONE`

For an exact match, the state match indicator only becomes true when the match value equals the channel-read value.

For an any match, a match is when the match value equals the channel-read value.

For an unchanged match, the value should be the same as the original. If not, a match is declared.

When none is set, matching is disabled.

This command is not supported on DAC and switch channels.

Example

```
channel.setmatchtype("121", channel.MATCH_EXACT)
```

Assuming a 7706 is installed, set the match type for digital I/O channel 21 on slot 1 to `MATCH_EXACT`.

Also see

[channel.getmatchtype\(\)](#) (on page 14-77)

[channel.setmatch\(\)](#) (on page 14-100)

channel.setmode()

This function sets the mode of operation of a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Recall setup Power cycle	Create configuration script Save setup	Digital I/O: channel.MODE_INPUT Totalizer: channel.MODE_RISING_EDGE

Usage

```
channel.setmode("channelList", mode)
```

<i>channelList</i>	String specifying the channels to set, using normal channel list syntax
<i>mode</i>	The mode of operation for a channel; for digital channels, the options are: - channel.MODE_INPUT - channel.MODE_OUTPUT For totalizer channels, the options are: - channel.MODE_RISING_EDGE - channel.MODE_FALLING_EDGE - channel.MODE_RISING_EDGE_READ_RESET - channel.MODE_FALLING_EDGE_READ_RESET

Details

You can set digital I/O channels to be input or output channels. Changing the mode from input to output adds an additional channel delay if a channel delay is set.

For totalizer channels, you can set which side of the input signal to count on and whether or not to reset the count when the count is read. To clear the totalizer count at a specific step in a scan, set the mode to either rising reset or falling reset, then use the add scan command to add the totalizer channel.

The specified channel list must define only one channel type. For example, channel list "121:125" is only valid if channels 21, 22, 23, 24, and 25 are the same type. If one of the channels is a different type of channel, the channel list is invalid and an error is generated.

Changing the mode setting can impact the power consumption of the card. The instrument verifies that power is available before changing the mode. If there is not enough power, the command generates an error.

This command is not available for DAC, switch, and backplane channels.

Example

```
channel.setmode("125", channel.MODE_FALLING_EDGE)
print(channel.getmode("125"))
```

Assuming a 7706, set totalizer 25 on slot 1 to be configured to count on the falling edge.
Print the setting to verify it. The return is:
[1]=0

Also see

[channel.getmode\(\)](#) (on page 14-78)
[channel.setdelay\(\)](#) (on page 14-87)
[scan.add\(\)](#) (on page 14-374)

channel.setwidth()

This function allows you to control multiple digital I/O channels as one channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.setwidth("channelNumber", width)
```

<i>channelNumber</i>	The first channel number of the channels to be combined
<i>width</i>	The number of channels to combine

Details

Changing the width allows for multiple sequential digital I/O channels to be linked. This option only applies to digital I/O channels, such as channels 21 and 22 on the 7706 and channels 11 to 14 on the 7707.

For widths greater than 1, the specified channel occupies the least significant byte. For example, writing the value of 4278255360 (hexadecimal FF00FF00) to channel 1 with a width of 4 sets channel 1 to 0, channel 2 to 255 (hexadecimal FF), channel 3 to 0, and channel 4 to 255 (hexadecimal FF). Writing the value of 4278255360 to channel 1 with a width of 1 sets channel 1 to 0 and leaves other channels untouched.

You must use decimal values when sending commands to the DAQ6510.

The maximum width is the number of digital I/O channels available on the card. If a width crosses channels, then only the channels set to output are affected.

Example

```
channel.setwidth("121", 2)
```

Assuming a 7706 in slot 1, sets the digital I/O channels starting at channel 21 to be handled as one channel.

Also see

[channel.getwidth\(\)](#) (on page 14-81)

channel.write()

This function writes a value to a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.write("channelList", value)
```

<i>channelList</i>	The channels to set, using standard channel naming (on page 11-6)
<i>value</i>	The value to be written to the channel (must be decimal value)

Details

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.

For DAC channels, the value is expected to be the set 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.

For digital I/O channels, the value becomes the setting of the digital output.

The time it takes to execute the write command is affected by the channel delay setting.

Example

<code>channel.write("121", 33)</code>	Output a value of 33 to digital I/O channel 1.

Also see

None

createconfigscript()

This function creates a setup file that captures most of the present settings of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
createconfigscript ("scriptName")
```

scriptName	A string that represents the name of the script to be created

Details

This function does not automatically overwrite existing scripts with the same name. If *scriptName* is set to the name of an existing script, an event message is returned. You must delete the existing script before using the same script name. This includes the `autoexec` script, which runs automatically when the instrument power is turned on. You can set *scriptName* to `autoexec`, but you must delete the existing `autoexec` script first using the `script.delete("autoexec")` command.

Once created, the script that contains the settings can be run and edited like any other script.

NOTE

Settings made on the Graph and Histogram tabs are not saved as part of a saved setup. To record graph settings, you can press HOME and ENTER to save an image of the settings with the screen capture feature. Refer to [Save screen captures to a USB flash drive](#) (on page 3-79) for additional information.

Example

```
createconfigscript ("myConfigurationScript")
reset ()
myConfigurationScript ()
```

Capture the present settings of the instrument into a script named `myConfigurationScript`.
Reset the instrument.
Restore the settings stored in `myConfigurationScript`.

Also see

- [Saving setups](#) (on page 4-92)
- [script.delete\(\)](#) (on page 14-405)

dataqueue.add()

This function adds an entry to the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
result = dataqueue.add(value)
result = dataqueue.add(value, timeout)
```

result	The resulting value of true or false based on the success of the function
value	The data item to add; <i>value</i> can be of any type
timeout	The maximum number of seconds to wait for space in the data queue

Details

You cannot use the *timeout* value when accessing the data queue from a remote node (you can only use the *timeout* value while adding data to the local data queue).

The *timeout* value is ignored if the data queue is not full.

The `dataqueue.add()` function returns *false*:

- If the timeout expires before space is available in the data queue
- If the data queue is full and a *timeout* value is not specified

If the value is a table, a duplicate of the table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
dataqueue.add(10)
dataqueue.add(11, 2)
result = dataqueue.add(12, 3)
if result == false then
    print("Failed to add 12 to the dataqueue")
end
print("The dataqueue contains:")
while dataqueue.count > 0 do
    print(dataqueue.next())
end
```

Clear the data queue.
Each line adds one item to the data queue.

Output:
The dataqueue contains:
10
11
12

Also see

[dataqueue.CAPACITY](#) (on page 14-107)
[dataqueue.clear\(\)](#) (on page 14-108)

[dataqueue.count](#) (on page 14-109)
[dataqueue.next\(\)](#) (on page 14-110)
[Using the data queue for real-time communications](#) (on page 9-14)

dataqueue.CAPACITY

This constant is the maximum number of entries that you can store in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
count = dataqueue.CAPACITY
```

count	The variable that is assigned the value of dataqueue.CAPACITY

Details

This constant always returns the maximum number of entries that can be stored in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
    dataqueue.add(1)
end
print("There are " .. dataqueue.count .. " items in the data queue")
```

This example fills the data queue until it is full and prints the number of items in the queue.

Output:

There are 128 items in the data queue

Also see

[dataqueue.add\(\)](#) (on page 14-106)
[dataqueue.clear\(\)](#) (on page 14-108)
[dataqueue.count](#) (on page 14-109)
[dataqueue.next\(\)](#) (on page 14-110)
[Using the data queue for real-time communications](#) (on page 9-14)

dataqueue.clear()

This function clears the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dataqueue.clear()
```

Details

This function forces all `dataqueue.add()` commands that are in progress to time out and deletes all data from the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
    dataqueue.add(1)
end
print("There are " .. dataqueue.count
    .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
    .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.

Output:

```
There are 128 items in the data
queue
```

```
There are 0 items in the data queue
```

Also see

[dataqueue.add\(\)](#) (on page 14-106)

[dataqueue.CAPACITY](#) (on page 14-107)

[dataqueue.count](#) (on page 14-109)

[dataqueue.next\(\)](#) (on page 14-110)

[Using the data queue for real-time communications](#) (on page 9-14)

dataqueue.count

This attribute contains the number of items in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
count = dataqueue.count
```

count	The number of items in the data queue

Details

The count is updated as entries are added with `dataqueue.add()` and read from the data queue with `dataqueue.next()`. It is also updated when the data queue is cleared with `dataqueue.clear()`.

A maximum of `dataqueue.CAPACITY` items can be stored at any one time in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
    dataqueue.add(1)
end
print("There are " .. dataqueue.count
    .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
    .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.

Output:

```
There are 128 items in the data queue
There are 0 items in the data queue
```

Also see

[dataqueue.add\(\)](#) (on page 14-106)

[dataqueue.CAPACITY](#) (on page 14-107)

[dataqueue.clear\(\)](#) (on page 14-108)

[dataqueue.next\(\)](#) (on page 14-110)

[Using the data queue for real-time communications](#) (on page 9-14)

dataqueue.next()

This function removes the next entry from the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = dataqueue.next()  
value = dataqueue.next(timeout)
```

value	The next entry in the data queue
timeout	The number of seconds to wait for data in the queue

Details

If the data queue is empty, the function waits up to the *timeout* value.

If data is not available in the data queue before the *timeout* expires, the return value is *nil*.

The entries in the data queue are removed in first-in, first-out (FIFO) order.

If the value is a table, a duplicate of the original table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
for i = 1, 10 do
    dataqueue.add(i)
end
print("There are " .. dataqueue.count .. " items in the data queue")

while dataqueue.count > 0 do
    x = dataqueue.next()
    print(x)
end
print("There are " .. dataqueue.count .. " items in the data queue")
```

Clears the data queue, adds ten entries, then reads the entries from the data queue. Note that your output may differ depending on the setting of `format.asciiprecision`.

Output:

```
There are 10 items in the data queue
1
2
3
4
5
6
7
8
9
10
There are 0 items in the data queue
```

Also see

[dataqueue.add\(\)](#) (on page 14-106)

[dataqueue.CAPACITY](#) (on page 14-107)

[dataqueue.clear\(\)](#) (on page 14-108)

[dataqueue.count](#) (on page 14-109)

[format.asciiprecision](#) (on page 14-341)

[Using the data queue for real-time communications](#) (on page 9-14)

delay()

This function delays the execution of the commands that follow it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

delay(*seconds*)

<i>seconds</i>	The number of seconds to delay (0 s to 100 ks)

Details

The instrument delays execution of the commands for at least the specified number of seconds and fractional seconds. However, the processing time may cause the instrument to delay 5 μ s to 10 μ s (typical) more than the requested delay.

Example2

```
beeper.beep(0.5, 2400)
delay(0.250)
beeper.beep(0.5, 2400)
```

Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Example 2

```
dataqueue.clear()
dataqueue.add(35)
timer.cleartime()
delay(0.5)
dt = timer.gettime()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it.
Output:
Delay time was 0.500099
35

Also see

None

digio.line[N].mode

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This attribute sets the mode of the digital I/O line to be a digital line, trigger line, or synchronous line and sets the line to be input, output, or open-drain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	digio.MODE_DIGITAL_IN

Usage

```
lineMode = digio.line[N].mode
digio.line[N].mode = lineMode
```

<i>lineMode</i>	The digital line control type and line mode: Digital control, input: digio.MODE_DIGITAL_IN Digital control, output: digio.MODE_DIGITAL_OUT Digital control, open-drain: digio.MODE_DIGITAL_OPEN_DRAIN Trigger control, input: digio.MODE_TRIGGER_IN Trigger control, output: digio.MODE_TRIGGER_OUT Trigger control, open-drain: digio.MODE_TRIGGER_OPEN_DRAIN Synchronous master: digio.MODE_SYNCHRONOUS_MASTER Synchronous acceptor: digio.MODE_SYNCHRONOUS_ACCEPTOR
<i>N</i>	The digital I/O line: 1 to 6

Details

You can use this command to place each digital I/O line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or synchronous acceptor

A digital line allows direct control of the digital I/O lines by writing a bit pattern to the lines. A trigger line uses the digital I/O lines to detect triggers.

The following settings of *lineMode* set the line for direct control as a digital line:

- `digio.MODE_DIGITAL_IN`: The instrument automatically detects externally generated logic levels. You can read an input line, but you cannot write to it.
- `digio.MODE_DIGITAL_OUT`: You can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low.
- `digio.MODE_DIGITAL_OPEN_DRAIN`: Configures the line to be an open-drain signal. The line can serve as an input, an output or both. When a digital I/O line is used as an input in open-drain mode, you must write a 1 to it.

The following settings of *lineMode* set the line as a trigger line:

- `digio.MODE_TRIGGER_IN`: The line automatically responds to and detects externally generated triggers. It detects falling-edge, rising-edge, or either-edge triggers as input. This line state uses the edge setting specified by the `trigger.digin[N].edge` attribute.
- `digio.MODE_TRIGGER_OUT`: The line is automatically set high or low depending on the output logic setting. Use the negative logic setting when you want to generate a falling edge trigger and use the positive logic setting when you want to generate a rising edge trigger.
- `digio.MODE_TRIGGER_OPEN_DRAIN`: Configures the line to be an open-drain signal. You can use the line to detect input triggers or generate output triggers. This line state uses the edge setting specified by the `trigger.digin[N].edge` attribute.

When the line is set as a synchronous acceptor, the line detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserting an output trigger releases the latched line.

When the line is set as a synchronous master, the line detects rising-edge triggers as input. For output, the line asserts a TTL-low pulse.

Example

```
digio.line[1].mode = digio.MODE_TRIGGER_OUT
```

Set digital I/O line 1 to be an output trigger line.

Also see

[Digital I/O lines](#) (on page 8-5)

[Digital I/O port configuration](#) (on page 8-3)

[trigger.digin\[N\].edge](#) (on page 14-449)

digio.line[N].reset()

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function resets digital I/O line values to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.line[N].reset()
```

<i>N</i>	The digital I/O line: 1 to 6

Details

This function resets the following attributes to their default values:

- `digio.line[N].mode`
- `trigger.digin[N].edge`
- `trigger.digout[N].logic`
- `trigger.digout[N].pulsewidth`
- `trigger.digout[N].stimulus`

It also clears `trigger.digin[N].overrun`.

Example

```
-- Set the digital I/O trigger line 3 for a falling edge
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.digout[3].logic = trigger.LOGIC_NEGATIVE
-- Set the digital I/O trigger line 3 to have a pulsewidth of 50 microseconds.
trigger.digout[3].pulsewidth = 50e-6
-- Use digital I/O line 5 to trigger the event on line 3.
trigger.digout[3].stimulus = trigger.EVENT_DIGIO5
-- Print configuration (before reset).
print(digio.line[3].mode, trigger.digout[3].pulsewidth,
      trigger.digout[3].stimulus)
-- Reset the line back to factory default values.
digio.line[3].reset()
-- Print configuration (after reset).
print(digio.line[3].mode, trigger.digout[3].pulsewidth,
      trigger.digout[3].stimulus)
```

Output before reset:

```
digio.MODE_TRIGGER_OUT    5e-05    trigger.EVENT_DIGIO5
```

Output after reset:

```
digio.MODE_TRIGGER_IN     1e-05    trigger.EVENT_NONE
```

Also see

[digio.line\[N\].mode](#) (on page 14-112)

[Digital I/O port configuration](#) (on page 8-3)

[trigger.digin\[N\].overrun](#) (on page 14-450)

[trigger.digout\[N\].pulsewidth](#) (on page 14-454)

[trigger.digout\[N\].stimulus](#) (on page 14-455)

digio.line[N].state

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function sets a digital I/O line high or low when the line is set for digital control and returns the state on the digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	See Details

Usage

```
digio.line[N].state = state
state = digio.line[N].state
```

<i>N</i>	The digital I/O line: 1 to 6
<i>state</i>	Set the line low: <code>digio.STATE_LOW</code> or 0 Set the line high: <code>digio.STATE_HIGH</code> or 1

Details

When a reset occurs, the digital line state can be read as high because the digital line is reset to a digital input. A digital input floats high if nothing is connected to the digital line.

This returns the integer equivalent values of the binary states on all six digital I/O lines.

Set the state to `digio.STATE_LOW` to clear the bit; set the state to `digio.STATE_HIGH` to set the bit.

Example

<pre>digio.line[1].mode = digio.MODE_DIGITAL_OUT digio.line[1].state = digio.STATE_HIGH</pre>	Sets line 1 (bit B1) of the digital I/O port high.

Also see

[digio.line\[N\].mode](#) (on page 14-112)
[digio.readport\(\)](#) (on page 14-116)
[digio.writeport\(\)](#) (on page 14-117)
[Digital I/O port configuration](#) (on page 8-3)
[trigger.digin\[N\].edge](#) (on page 14-449)

digio.readport()

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function reads the digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = digio.readport()
```

<i>data</i>	The present value of the input lines on the digital I/O port

Details

The binary equivalent of the returned value indicates the value of the input lines on the digital I/O port. The least significant bit (bit B1) of the binary number corresponds to digital I/O line 1; bit B6 corresponds to digital I/O line 6.

For example, a returned value of 42 has a binary equivalent of 101010, which indicates that lines 2, 4, 6 are high (1), and the other lines are low (0).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

Example

```
data = digio.readport()
print(data)
```

Assume lines 2, 4, and 6 are set high when the I/O port is read.
Output:
42
This is binary 101010.

Also see

[digio.writeport\(\)](#) (on page 14-117)

[Digital I/O port configuration](#) (on page 8-3)

digio.writeport()

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function writes to all digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.writeport(data)
```

<i>data</i>	The value to write to the port (0 to 63)

Details

This function writes to the digital I/O port by setting the binary state of each digital line from an integer equivalent value.

The binary representation of the value indicates the output pattern to be written to the I/O port. For example, a value of 63 has a binary equivalent of 111111 (all lines are set high); a *data* value of 42 has a binary equivalent of 101010 (lines 2, 4, and 6 are set high, and the other three lines are set low).

An instrument reset does not affect the present states of the digital I/O lines.

All six lines must be configured as digital control lines. If not, this command generates an error.

You can also enter the data parameter as a binary value.

Example 1

<code>digio.writeport(63)</code>	Sets digital I/O lines 1 through 6 high (binary 111111).

Example 2

<code>digio.writeport(0b111111)</code>	Sets digital I/O lines 1 through 6 high (digital 63).

Also see

- [digio.readport\(\)](#) (on page 14-116)
- [Digital I/O port configuration](#) (on page 8-3)
- [Outputting a bit pattern](#) (on page 8-13)

display.activebuffer

This attribute determines which buffer is used for measurements that are displayed on the front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	defbuffer1

Usage

```
bufferName = display.activebuffer
display.activebuffer = bufferName
```

<i>bufferName</i>	The name of the buffer to make active

Details

The buffer defined by this command is used to store measurements data and is shown in the reading buffer indicator on the home screen of the instrument.

Example

```
display.activebuffer = buffer2
```

Set the front panel to use `buffer2` as the active reading buffer.

Also see

None

display.changescreen()

This function changes which front-panel screen is displayed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`display.changescreen(screenName)`

<i>screenName</i>	The screen to display: ■ Home screen: <code>display.SCREEN_HOME</code> ■ Home screen with large readings: <code>display.SCREEN_HOME_LARGE_READING</code> ■ Reading table screen: <code>display.SCREEN_READING_TABLE</code> ■ Graph screen (opens last selected tab): <code>display.SCREEN_GRAPH</code> ■ Histogram: <code>display.SCREEN_HISTOGRAM</code> ■ FUNCTIONS swipe screen: <code>display.SCREEN_FUNCTIONS_SWIPE</code> ■ GRAPH swipe screen: <code>display.SCREEN_GRAPH_SWIPE</code> ■ SECONDARY swipe screen: <code>display.SCREEN_SECONDARY_SWIPE</code> ■ SETTINGS swipe screen: <code>display.SCREEN_SETTINGS_SWIPE</code> ■ STATISTICS swipe screen: <code>display.SCREEN_STATS_SWIPE</code> ■ USER swipe screen: <code>display.SCREEN_USER_SWIPE</code> (only displays USER swipe screen if user text is sent) ■ CHANNEL swipe screen: <code>display.SCREEN_CHANNEL_SWIPE</code> (only available when a card is installed and rear terminals are selected) ■ NONSWITCH swipe screen: <code>display.SCREEN_NONSWITCH_SWIPE</code> (only available when a card with nonswitching channels is installed and the rear terminals are selected) ■ SCAN swipe screen: <code>display.SCREEN_SCAN_SWIPE</code> (only available when a card is installed and the rear terminals are selected) ■ Channel control screen: <code>display.SCREEN_CHANNEL_CONTROL</code> ■ Channel settings screen: <code>display.SCREEN_CHANNEL_SETTINGS</code> ■ Channel scan screen: <code>display.SCREEN_CHANNEL_SCAN</code> ■ Open a screen that uses minimal CPU resources: <code>display.SCREEN_PROCESSING</code>

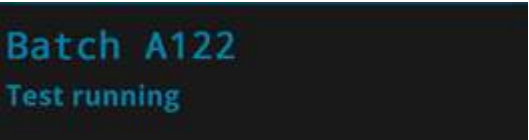
Details

The scan and channel options are only available if you have a card installed and if the front-panel TERMINALS switch is set to REAR. The nonswitch option is only available for cards such as the 7706 and 7707 that have nonswitching channels.

Example

```
display.clear()
display.settext(display.TEXT1, "Batch A122")
display.settext(display.TEXT2, "Test running")
display.changescreen(display.SCREEN_USER_SWIPE)
```

Clear the USER swipe screen.
Set the first line of the USER swipe screen to read "Batch A122" and the second line to display "Test running".
Display the USER swipe screen.



Also see

[display.settext\(\)](#) (on page 14-133)

display.clear()

This function clears the text from the front-panel USER swipe screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.clear()
```

Example

<pre>display.clear() display.changescreen(display.SCREEN_USER_SWIPE) display.settext(display.TEXT1, "Serial number:") display.settext(display.TEXT2, localnode.serialno)</pre>	Clear the USER swipe screen. Set the first line to read "Serial number:" and the second line to display the serial number of the instrument.

Also see

[display.settext\(\)](#) (on page 14-133)

display.delete()

This function allows you to remove a prompt on the front-panel display that was created with `display.prompt()`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.delete(promptID)
```

<code>promptID</code>	The identifier defined by <code>display.prompt()</code>

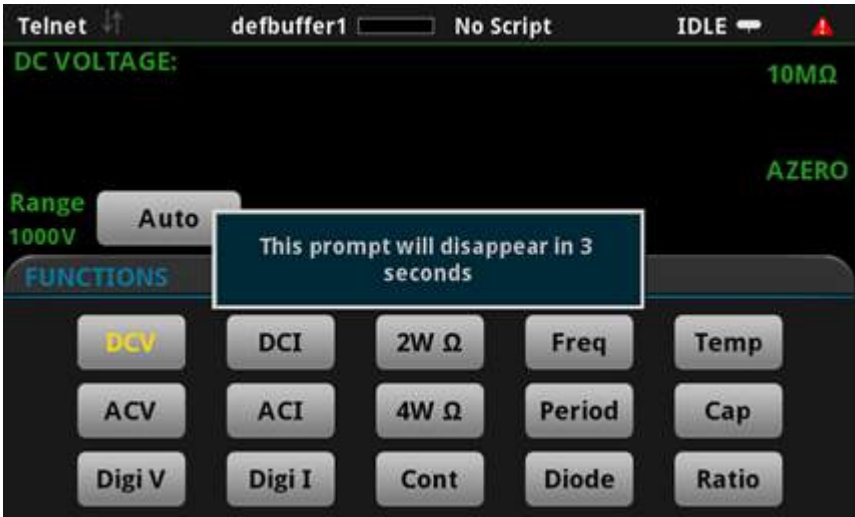
Details

You can use this command to remove the presently displayed prompt.

Example

```
removePrompt3 = display.prompt(display.BUTTONS_NONE, "This prompt will disappear  
in 3 seconds")  
delay(3)  
display.delete(removePrompt3)
```

This example displays a prompt that is automatically removed in three seconds.



Also see

[display.prompt\(\)](#) (on page 14-131)

display.input.number()

This function allows you to create a prompt that requests a number from the user on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
numberEntered = display.input.number("dialogTitle")
numberEntered = display.input.number("dialogTitle", numberFormat)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue,
    minimumValue)
numberEntered = display.input.number("dialogTitle", numberFormat, defaultValue,
    minimumValue, maximumValue)
```

<i>numberEntered</i>	The number that is entered from the front-panel display; nil if Cancel is pressed on the keypad
<i>dialogTitle</i>	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; can be up to 32 characters
<i>numberFormat</i>	The format of the displayed number: - Allow integers (negative or positive) only: <code>display.NFORMAT_INTEGER</code> (default) - Allow decimal values: <code>display.NFORMAT_DECIMAL</code> - Display numbers in exponent format: <code>display.NFORMAT_EXPONENT</code> - Display numbers with prefixes before the units symbol, such as n, m, or μ: <code>display.NFORMAT_PREFIX</code>
<i>defaultValue</i>	The value that is initially displayed in the displayed keypad
<i>minimumValue</i>	The lowest value that can be entered
<i>maximumValue</i>	The highest value that can be entered

Details

This command prompts the instrument operator to enter a value.

The prompt is displayed until it has been responded to.

NOTE

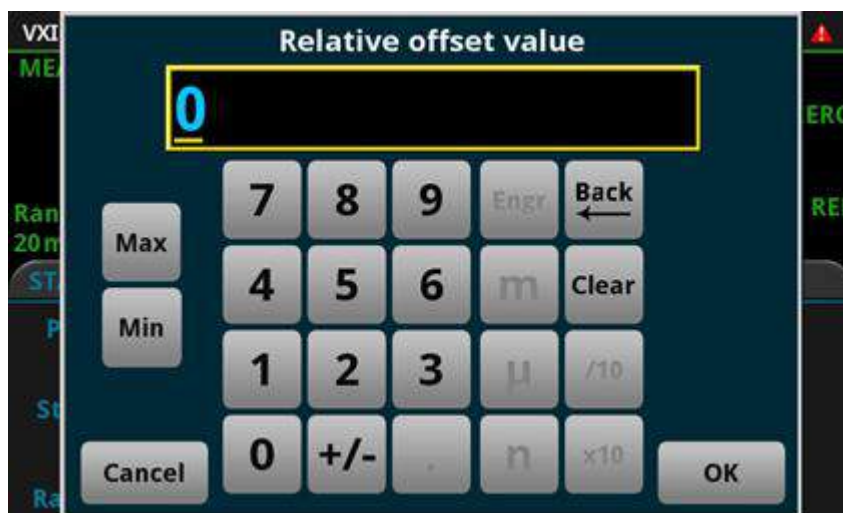
On the prompt, the operator can move the cursor in the entry box by touching the screen. The cursor is moved to the spot where the operator touched the screen.

Example

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.rel.enable = dmm.ON
relativeoffset = display.input.number("Relative offset value",
    display.NFORMAT_INTEGER, 0, -1000, 1000)
dmm.measure.rel.level = relativeoffset
```

This example displays a number pad on the screen that defaults to 0 and allows entries from –1000 to 1000. The number that the operator enters is assigned to the relative offset level. If the operator enters a value outside of the range, an error message is displayed.

Figure 176: Input number example



Also see

[display.input.option\(\)](#) (on page 14-125)

[display.input.prompt\(\)](#) (on page 14-127)

[display.input.string\(\)](#) (on page 14-128)

display.input.option()

This function allows you to create an option dialog box with customizable buttons on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.BUTTON_OPTIONn = display.input.option("dialogTitle", "buttonTitle1",
"buttonTitle2")
display.BUTTON_OPTIONn = display.input.option("dialogTitle", "buttonTitle1",
"buttonTitle2", "buttonTitleN", ... "buttonTitleN")
```

<i>n</i>	The number of the button that is selected from the front-panel display; nil if Cancel is pressed on the keypad; buttons are numbered top to bottom, left to right
<i>dialogTitle</i>	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 32 characters
<i>buttonTitle1</i>	A string that contains the name of the first button; up to 15 characters
<i>buttonTitle2</i>	A string that contains the name of the second button; up to 15 characters
<i>buttonTitleN</i>	A string that contains the names of subsequent buttons, where <i>N</i> is a number from 3 to 10; you can define up to 10 buttons; each button can be up to 15 characters

Details

Buttons are created from top to bottom, left to right. If you have more than five buttons, they are placed into two columns.

The prompt is displayed until it has been responded to. You can only send one input prompt command at a time.

Example

```
optionID = display.input.option("Select an option", "Apple", "Orange", "Papaya",  
    "Pineapple", "Blueberry", "Banana", "Grapes", "Peach", "Apricot", "Guava")  
print(optionID)
```

This example displays the following dialog box:



If the user selects `Peach`, the return is `display.BUTTON_OPTION8`.

Also see

[display.input.number\(\)](#) (on page 14-123)

[display.input.prompt\(\)](#) (on page 14-127)

[display.input.string\(\)](#) (on page 14-128)

display.input.prompt()

This function allows you to create a prompt that accepts a user response from the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
buttonReturn = display.input.prompt(buttonSet, "dialogTitle")
```

<i>buttonReturn</i>	Indicates which button was pressed: ■ OK: <code>display.BUTTON_OK</code> ■ Cancel: <code>display.BUTTON_CANCEL</code> ■ Yes: <code>display.BUTTON_YES</code> ■ No: <code>display.BUTTON_NO</code>
<i>buttonSet</i>	The set of buttons to display: ■ OK button only: <code>display.BUTTONS_OK</code> ■ Cancel button only: <code>display.BUTTONS_CANCEL</code> ■ OK and Cancel buttons: <code>display.BUTTONS_OKCANCEL</code> ■ Yes and No buttons: <code>display.BUTTONS_YESNO</code> ■ Yes, No, and Cancel buttons: <code>display.BUTTONS_YESNOCANCEL</code>
<i>dialogTitle</i>	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 63 characters

Details

This command waits for a user response to the prompt. You can use the text to ask questions that can be used to configure your test.

The prompt is displayed until it has been responded to by the user. You can only send one input prompt command at a time.

Example

```
result = display.input.prompt(display.BUTTONS_YESNO, "Do you want to display the
graph screen?")
if result == display.BUTTON_YES then
    display.changescreen(display.SCREEN_GRAPH)
end
```

This displays the prompt "Do you want to display the graph screen?" on the front-panel display:



If the operator selects **Yes**, the graph screen is displayed.

Also see

- [display.input.number\(\)](#) (on page 14-123)
- [display.input.option\(\)](#) (on page 14-125)
- [display.input.string\(\)](#) (on page 14-128)

display.input.string()

This function allows you to create a dialog box that requests text from the user through the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
textEntered = display.input.string("dialogTitle")
textEntered = display.input.string("dialogTitle", textFormat)
```

<i>textEntered</i>	The text that is entered from the front-panel display; nil if Cancel is pressed on the keypad
<i>dialogTitle</i>	A string that contains the text to be displayed as the title of the dialog box on the front-panel display; up to 32 characters
<i>textFormat</i>	The format of the entered text: - Allow any characters: <code>display.SFORMAT_ANY</code> (default) - Allow both upper- and lower-case letters (no special characters): <code>display.SFORMAT_UPPER_LOWER</code> - Allow only upper-case letters: <code>display.SFORMAT_UPPER</code> - Allow both upper- and lower-case letters, no special characters, no spaces, and limited to 32 characters: <code>display.SFORMAT_BUFFER_NAME</code>

Details

This command creates a prompt to the instrument operator to enter a string value.

The prompt is displayed until it has been responded to. You can only send one input prompt command at a time.

Example

```
value = display.input.string("Enter Test Name", display.SFORMAT_ANY)
print(value)
```

This example displays the prompt "Enter Test Name" and a keyboard that the operator can use to enter a response.



The return is the response from the operator.

Also see

[display.input.number\(\)](#) (on page 14-123)

[display.input.option\(\)](#) (on page 14-125)

[display.input.prompt\(\)](#) (on page 14-127)

display.lightstate

This attribute sets the light output level of the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	display.STATE_LCD_50

Usage

```
brightness = display.lightstate
display.lightstate = brightness
```

brightness

The brightness of the display:

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

Details

This command changes the light output of the front panel when a test requires different instrument illumination levels.

The change in illumination is temporary. The normal backlight settings are restored after a power cycle. You can use this to reset a display that is already dimmed by the front-panel Backlight Dimmer.

NOTE

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

Example

```
display.lightstate = display.STATE_LCD_50
```

Set the display brightness to 50%.

Also see

[Adjust the backlight brightness and dimmer](#) (on page 3-6)

display.prompt()

This function allows you to create an interactive dialog prompt that displays a custom message on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
promptID = display.prompt(buttonID, "promptText")
```

<i>promptID</i>	A set of characters that identifies the prompt; up to 63 characters
<i>buttonID</i>	The type of prompt to display; choose one of the following options: - display.BUTTONS_NONE - display.BUTTONS_OK - display.BUTTONS_CANCEL - display.BUTTONS_OKCANCEL - display.BUTTONS_YESNO - display.BUTTONS_YESNOCANCEL
<i>promptText</i>	A string that contains the text that is displayed above the prompts

Details

This command displays buttons and text on the front panel. You can set up scripts that respond to the buttons when they are selected.

If you send `display.BUTTONS_NONE`, the operator needs to press the EXIT key to clear the message from the front-panel display. You can also use the `display.delete()` command to remove the prompt.

Only one prompt can be active at a time.

When the user presses a button, the button presses are returned as one of the following options:

- OK: `display.BUTTON_OK`
- Cancel: `display.BUTTON_CANCEL`
- Yes: `display.BUTTON_YES`
- No: `display.BUTTON_NO`

To capture return values, you need to use `display.waitevent()` to wait for the user button selection.

Example

```
reset()
trigger.model.load("SimpleLoop", 10, 0, defbuffer1)
display.prompt(display.BUTTONS_YESNO, "Would you like to make 10 DC voltage
  readings now?")
promptID, result = display.waitevent()
if result == display.BUTTON_YES then
  trigger.model.initiate()
end
display.prompt(display.BUTTONS_YESNO, "Would you like to switch to the Graph
  screen?")
promptID, result = display.waitevent()
if result == display.BUTTON_YES then
  display.changescreen(display.SCREEN_GRAPH)
end
```

Create a simple loop that will make 10 measurements and save them in default buffer 1. Display the prompt shown here:



If the user presses Yes, the measurements are made.
If the user presses No, the measurements are not made, and the message is removed.
Display the prompt "Would you like to switch to the Graph screen?"
If the user presses Yes, the Graph screen is displayed.
If the user presses No, the user remains on the present screen.

Also see

[display.delete\(\)](#) (on page 14-122)

[display.waitevent\(\)](#) (on page 14-135)

display.readingformat

This attribute determines the format that is used to display measurement readings on the front-panel display of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	display.FORMAT_PREFIX

Usage

```
format = display.readingformat
display.readingformat = format
```

<i>format</i>	Use exponent format: <code>display.FORMAT_EXPONENT</code> Add a prefix to the units symbol, such as k, m, or μ : <code>display.FORMAT_PREFIX</code>

Details

This setting persists through `reset()` and power cycles.

When Prefix is selected, prefixes are added to the units symbol, such as k (kilo) or m (milli). When Exponent is selected, exponents are used instead of prefixes. When the prefix option is selected, very large or very small numbers may be displayed with exponents.

Example

```
display.readingformat = display.FORMAT_EXPONENT
```

Change front-panel display to show readings in exponential format.

Also see

[Setting the display format](#) (on page 3-76)

display.settext()

This function defines the text that is displayed on the front-panel USER swipe screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.settext(display.TEXT1, "userDisplayText1")
display.settext(display.TEXT2, "userDisplayText2")
```

<i>userDisplayText1</i>	String that contains the message for the top line of the USER swipe screen (up to 20 characters)
<i>userDisplayText2</i>	String that contains the message for the bottom line of the USER swipe screen (up to 32 characters)

Details

This command defines text messages for the USER swipe screen.

If you enter too many characters, the instrument displays a warning event and shortens the message to fit.

When the instrument is reset, the user test is removed and the USER swipe screen is hidden until another message is defined.

You can use the following codes to create special characters in the message.

Code	Special character (description)
\018	Ω (ohm)
\019	° (degree symbol)
\020	μ (mu)
\021	Thin space (less than i)
\178	² (superscript 2, for example, x ²)
\179	³ (superscript 3, for example, x ³)
\185	Δ (delta)
\188	1/x (reciprocal)
\189	Ratio

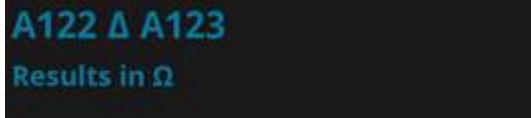
Example

```
display.clear()
display.changescreen(display.SCREEN_USER_SWIPE)
display.settext(display.TEXT1, "A122 \185 A123")
display.settext(display.TEXT2, "Results in \018")
```

Clear the USER swipe screen.

Display the USER swipe screen.

Set the first line to read "A122 Δ A123" and the second line to display "Results in Ω ":



A122 Δ A123
Results in Ω

Also see

[display.clear\(\)](#) (on page 14-121)

[display.changescreen\(\)](#) (on page 14-120)

display.waitevent()

This function causes the instrument to wait for a user to respond to a prompt that was created with a prompt command.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
objectID, subID = display.waitevent()  
objectID, subID = display.waitevent(timeout)
```

objectID	A number that identifies the object, such as a prompt message, that is displayed on the front panel
subID	The returned value after a button is pressed on the front panel: - display.BUTTON_YES - display.BUTTON_NO - display.BUTTON_OK - display.BUTTON_CANCEL
timeout	The amount of time to wait before timing out; time is 0 to 300 s, where the default of 0 waits indefinitely

Details

This command waits until a user responds to a front-panel prompt that was created with the display.prompt() command.

Example

```
reset()  
trigger.model.load("SimpleLoop", 10, 0, defbuffer1)  
display.prompt(display.BUTTONS_YESNO, "Would you like to make 10 DC voltage  
readings now?")  
promptID, result = display.waitevent()  
if result == display.BUTTON_YES then  
    trigger.model.initiate()  
end  
display.prompt(display.BUTTONS_YESNO, "Would you like to switch to the Graph  
screen?")  
promptID, result = display.waitevent()  
if result == display.BUTTON_YES then  
    display.changescreen(display.SCREEN_GRAPH)  
end
```

Create a simple loop that will make 10 measurements and save them in default buffer 1. Display the prompt "Would you like to make 10 DC voltage readings now?" If the user presses Yes, the measurements are made. If the user presses No, the measurements are not made and the message is removed. Display the prompt "Would you like to switch to the Graph screen?" If the user presses Yes, the Graph screen is displayed. If the user presses No, the user remains on the present screen.

Also see

[display.input.prompt\(\)](#) (on page 14-127)

[display.prompt\(\)](#) (on page 14-131)

display.watchchannels

This attribute determines which channels are set to be watch channels on the front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	None

Usage

```
"channelList" = display.watchchannels
display.watchchannels = "channelList"
```

<code>channelList</code>	The channels to set, using standard channel naming (on page 14-3)

Details

Watch channels are channels that you want to focus attention on. Watch channels affect what you see on the CHANNEL and STATISTICS swipe screens. They also determine which readings you see on the home screen.

In the Reading Table, you can select the watch channels to filter the buffer so that only information from the watched channels is shown.

In the Graph screens, you can select the watch channels. Each channel in the watch channels list is displayed as a trace on the graph.

You can define up to 20 channels as watch channels.

Example

```
display.watchchannels = "101:105"
```

Sets the instrument to watch channels 1, 2, 3, 4, and 5.

Also see

None

dmm.digitize.analogtrigger.edge.level

This attribute defines the signal level that generates the analog trigger event for the edge trigger mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0

Usage

```
value = dmm.digitize.analogtrigger.edge.level
dmm.digitize.analogtrigger.edge.level = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_ATRIG_EDGE_LEVEL)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_ATRIG_EDGE_LEVEL, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_ATRIG_EDGE_LEVEL, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_EDGE_LEVEL)
```

<i>value</i>	The signal level that generates the analog trigger event
<i>function</i>	The digitize function to which to assign this parameter: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

This command is only available when the analog trigger mode is set to edge.

The edge level can be set to any value in the active measurement range.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.analogtrigger.mode = dmm.MODE_EDGE
dmm.digitize.analogtrigger.edge.level = 5
dmm.digitize.analogtrigger.edge.slope = dmm.SLOPE_FALLING
```

Set the function to digitize voltage.
Set the analog trigger mode to edge.
Set the level to sense 5 V.
Set the level to be detected on a falling edge.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_MODE, dmm.MODE_EDGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_EDGE_LEVEL, 5)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE, dmm.SLOPE_FALLING)
```

For channels 1 through 9 on slot 1, set the digitize function to voltage.
Set range to 10 V.
Set the analog trigger mode to edge.
Set the analog trigger level to 5 V.
Set the level to be detected on a falling edge.

Also see

- [Analog triggering overview](#) (on page 8-20)
- [dmm.digitize.analogtrigger.mode](#) (on page 14-140)
- [dmm.digitize.analogtrigger.edge.slope](#) (on page 14-138)
- [dmm.digitize.func](#) (on page 14-155)
- [dmm.measure.analogtrigger.edge.level](#) (on page 14-191)

dmm.digitize.analogtrigger.edge.slope

This attribute defines the slope of the analog trigger edge.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.SLOPE_RISING

Usage

```
value = dmm.digitize.analogtrigger.slope
dmm.digitize.analogtrigger.slope = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE)
```

value	The slope of the analog trigger edge: ■ Rising: dmm.SLOPE_RISING ■ Falling: dmm.SLOPE_FALLING
function	The digitize function to which to assign this parameter: ■ Current: dmm.FUNC_DIGITIZE_CURRENT ■ Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

This is only available when the analog trigger mode is set to edge.

Rising causes an analog trigger event when the analog signal trends from below the analog signal level to above the level.

Falling causes an analog trigger event when the signal trends from above to below the level.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.analogtrigger.mode = dmm.MODE_EDGE
dmm.digitize.analogtrigger.edge.level = 5
dmm.digitize.analogtrigger.edge.slope = dmm.SLOPE_FALLING
```

Set the function to digitize voltage.
Set the analog trigger mode to edge.
Set the level to sense 5 V.
Set the level to be detected on a falling edge.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_MODE, dmm.MODE_EDGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_EDGE_LEVEL, 5)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE, dmm.SLOPE_FALLING)
```

For channels 1 through 9 on slot 1, set the digitize function to voltage.

Set range to 10 V.

Set the analog trigger mode to edge.

Set the analog trigger level to 5 V.

Set the level to be detected on a falling edge.

Also see

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.edge.level](#) (on page 14-137)

[dmm.digitize.analogtrigger.mode](#) (on page 14-140)

[dmm.digitize.func](#) (on page 14-155)

[dmm.measure.analogtrigger.edge.slope](#) (on page 14-193)

dmm.digitize.analogtrigger.mode

This attribute configures the type of signal behavior that can generate an analog trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.MODE_OFF

Usage

```
setting = dmm.digitize.analogtrigger.mode
dmm.digitize.analogtrigger.mode = setting
setting = dmm.measure.getattribute(function, dmm.ATTR_DIGI_ATTRIG_MODE)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_ATTRIG_MODE, setting)
channel.setdmm("channellist", dmm.ATTR_DIGI_ATTRIG_MODE, setting)
setting = channel.getdmm("channellist", dmm.ATTR_DIGI_ATTRIG_MODE)
```

<i>setting</i>	The mode setting: ▪ Edge (signal crosses one level): dmm.MODE_EDGE ▪ Window (signal enters or exits a window defined by two levels): dmm.MODE_WINDOW ▪ No analog triggering: dmm.MODE_OFF
<i>function</i>	The digitize function to which to assign this parameter: ▪ Current: dmm.FUNC_DIGITIZE_CURRENT ▪ Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channellist</i>	The channels to set, using standard channel naming (on page 14-3)

Details

When edge is selected, the analog trigger occurs when the signal crosses a certain level. You also specify if the analog trigger occurs on the rising or falling edge of the signal.

When window is selected, the analog trigger occurs when the signal enters or exits the window defined by the low and high signal levels.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.range = 90
dmm.digitize.analogtrigger.mode = dmm.MODE_EDGE
dmm.digitize.analogtrigger.edge.level = 5
dmm.digitize.analogtrigger.edge.slope = dmm.SLOPE_FALLING
```

Set the function to digitize voltage.
Set the range to 90, which selects a range of 100 V.
Set the analog trigger mode to edge.
Set the level sense to 5 V.
Set the level to be detected on a falling edge.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_MODE, dmm.MODE_EDGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_EDGE_LEVEL, 5)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE, dmm.SLOPE_FALLING)
```

For channels 1 through 9 on slot 1, set the digitize function to voltage.
Set range to 10 V.
Set the analog trigger mode to edge.
Set the analog trigger level to 5 V.
Set the level to be detected on a falling edge.

Also see

[Analog triggering overview](#) (on page 8-20)
[dmm.measure.analogtrigger.mode](#) (on page 14-195)

dmm.digitize.analogtrigger.window.direction

This attribute defines if the analog trigger occurs when the signal enters or leaves the defined high and low analog signal level boundaries.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.DIRECTION_ENTER

Usage

```
value = dmm.digitize.analogtrigger.window.direction
dmm.digitize.analogtrigger.window.direction = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_WINDOW_DIRECTION)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_WINDOW_DIRECTION, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_ATRIG_WINDOW_DIRECTION, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_ATRIG_WINDOW_DIRECTION)
```

value	The direction: - Enter: dmm.DIRECTION_ENTER - Leave: dmm.DIRECTION_LEAVE
function	The digitize function to which to assign this parameter: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

This is only available when the analog trigger mode is set to window.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.analogtrigger.mode = dmm.MODE_WINDOW
dmm.digitize.analogtrigger.window.levelhigh = 5
dmm.digitize.analogtrigger.window.levellow = 1
dmm.digitize.analogtrigger.window.direction = dmm.DIRECTION_LEAVE
```

Set function to digitize voltage.
Set the analog trigger mode to window.
Set the analog trigger high level to 5 V.
Set the analog trigger low level to 1 V.
Set the trigger to occur when the signal leaves the window.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_MODE, dmm.MODE_WINDOW)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_HIGH, 5)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_LOW, 1)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_DIRECTION,
    dmm.DIRECTION_LEAVE)
```

For channels 1 through 9 on slot 1, set the digitize function to DC voltage.

Set range to 10 V.

Set the analog trigger mode to window.

Set the analog trigger high level to 5 V.

Set the analog trigger low level to 1 V.

Set the trigger to occur when the signal leaves the window.

Also see

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.mode](#) (on page 14-140)

[dmm.digitize.analogtrigger.window.levelhigh](#) (on page 14-144)

[dmm.digitize.analogtrigger.window.levellow](#) (on page 14-146)

[dmm.digitize.func](#) (on page 14-155)

[dmm.measure.analogtrigger.window.direction](#) (on page 14-197)

dmm.digitize.analogtrigger.window.levelhigh

This attribute defines the upper boundary of the analog trigger window.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Digitize current: 5e-6 (5 μ A) Digitize voltage: 0.05 (50 mV)

Usage

```
value = dmm.digitize.analogtrigger.window.levelhigh
dmm.digitize.analogtrigger.window.levelhigh = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_WINDOW_LEVEL_HIGH)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_WINDOW_LEVEL_HIGH, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_HIGH, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_HIGH)
```

<i>value</i>	The upper boundary of the window
<i>function</i>	The digitize function to which to assign this parameter: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

Only available when the analog trigger mode is set to window.

The high level must be greater than the low level.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.analogtrigger.mode = dmm.MODE_WINDOW
dmm.digitize.analogtrigger.window.levelhigh = 5
dmm.digitize.analogtrigger.window.levellow = 1
dmm.digitize.analogtrigger.window.direction = dmm.DIRECTION_LEAVE
```

Set function to digitize voltage.
Set the analog trigger mode to window.
Set the analog trigger high level to 5 V.
Set the analog trigger low level to 1 V.
Set the trigger to occur when the signal leaves the window.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_MODE, dmm.MODE_WINDOW)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_HIGH, 5)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_LOW, 1)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_DIRECTION,
    dmm.DIRECTION_LEAVE)
```

For channels 1 through 9 on slot 1, set the digitize function to DC voltage.

Set range to 10 V.

Set the analog trigger mode to window.

Set the analog trigger high level to 5 V.

Set the analog trigger low level to 1 V.

Set the trigger to occur when the signal leaves the window.

Also see

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.mode](#) (on page 14-140)

[dmm.digitize.analogtrigger.window.direction](#) (on page 14-142)

[dmm.digitize.analogtrigger.window.levellow](#) (on page 14-146)

[dmm.digitize.func](#) (on page 14-155)

[dmm.measure.analogtrigger.window.levelhigh](#) (on page 14-199)

dmm.digitize.analogtrigger.window.levellow

This attribute defines the lower boundary of the analog trigger window.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0

Usage

```
value = dmm.digitize.analogtrigger.window.levellow
dmm.digitize.analogtrigger.window.levellow = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_WINDOW_LEVEL_LOW)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_WINDOW_LEVEL_LOW, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_LOW, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_LOW)
```

value	The lower boundary of the window
function	The digitize function to which to assign this parameter: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

Only available when the analog trigger mode is set to window.

The low level must be less than the high level.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.analogtrigger.mode = dmm.MODE_WINDOW
dmm.digitize.analogtrigger.window.levelhigh = 5
dmm.digitize.analogtrigger.window.levellow = 1
dmm.digitize.analogtrigger.window.direction = dmm.DIRECTION_LEAVE
```

Set function to digitize voltage.
Set the analog trigger mode to window.
Set the analog trigger high level to 5 V.
Set the analog trigger low level to 1 V.
Set the trigger to occur when the signal leaves the window.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_DIGI_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_MODE, dmm.MODE_WINDOW)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_HIGH, 5)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_LOW, 1)
channel.setdmm("101:109", dmm.ATTR_DIGI_ATRIG_WINDOW_DIRECTION,
    dmm.DIRECTION_LEAVE)
```

For channels 1 through 9 on slot 1, set the digitize function to DC voltage.

Set range to 10 V.

Set the analog trigger mode to window.

Set the analog trigger high level to 5 V.

Set the analog trigger low level to 1 V.

Set the trigger to occur when the signal leaves the window.

Also see

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.mode](#) (on page 14-140)

[dmm.digitize.analogtrigger.window.direction](#) (on page 14-142)

[dmm.digitize.analogtrigger.window.levelhigh](#) (on page 14-144)

[dmm.digitize.func](#) (on page 14-155)

[dmm.measure.analogtrigger.window.levellow](#) (on page 14-201)

dmm.digitize.aperture

This attribute determines the aperture setting for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.APERTURE_AUTO

Usage

```
time = dmm.digitize.aperture
dmm.digitize.aperture = time
time = dmm.measure.getattribute(function, dmm.ATTR_DIGI_APERTURE)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_APERTURE, time)
channel.setdmm("channelList", dmm.ATTR_DIGI_APERTURE, time)
time = channel.getdmm("channelList", dmm.ATTR_DIGI_FUNCTION)
```

<i>time</i>	The time of the aperture in seconds or automatic: - Range: 1 μs to 1 ms; set in 1 μs increments - Automatic: 0 or dmm.APERTURE_AUTO
<i>function</i>	The digitize function to which to assign this parameter: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

The aperture is the actual acquisition time of the instrument on the signal. The aperture can be set to automatic or to a specific value in 1 μ s intervals. If the value is not specified in microseconds, the value is rounded down to the nearest microsecond resolution. When automatic is selected, the aperture setting is set to the maximum value possible for the selected sample rate.

The aperture must be less than the reciprocal of the sample rate. The minimum aperture is 1 μ s at the maximum sampling rate of 1,000,000 samples per second.

Set the sample rate before changing the aperture.

The maximum aperture available is 1 divided by the sample rate. The aperture cannot be set to more than this value.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.samplerate = 200000
dmm.digitize.aperture = dmm.APERTURE_AUTO
dmm.digitize.count = 1
print(dmm.digitize.read())
```

Set the digitize function to measure current. Set the sample rate to 200,000, with a count of 1, and automatic aperture.
Make a digitize measurement.

Example 2

```
channel.setdmm("121:122", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_CURRENT)
channel.setdmm("121:122", dmm.ATTR_DIGI_SAMPLE_RATE, 200000,
    dmm.ATTR_DIGI_APERTURE, dmm.APERTURE_AUTO, dmm.ATTR_DIGI_COUNT, 1)
channel.close("121")
print(dmm.digitize.read())
channel.close("122")
print(dmm.digitize.read())
```

With a 7700 switching module installed, set the measurement function on channels 21 and 22 of slot 1 to digitize current, with a rate of 200,000, automatic aperture, and a count of 1.
Make digitize measurements on each of the channels.

Also see

[Digitize functions](#) (on page 4-35)
[dmm.digitize.func](#) (on page 14-155)
[dmm.digitize.samplerate](#) (on page 14-187)
[dmm.measure.aperture](#) (on page 14-203)

dmm.digitize.count

This attribute sets the number of measurements to digitize when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	10,000

Usage

```
count = dmm.digitize.count
dmm.digitize.count = count
count = dmm.measure.getattribute(function, dmm.ATTR_DIGI_COUNT)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_COUNT, count)
channel.setdmm("channelList", dmm.ATTR_DIGI_COUNT, count)
count = channel.getdmm("channelList", dmm.ATTR_DIGI_COUNT)
```

count	The number of measurements to make (1 to 55,000,000 or buffer capacity)
function	The digitize function to which to assign this parameter: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

The digitize function makes the number of readings set by this command in the time set by the sample rate. This command does not affect the trigger model.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.aperture = dmm.APERTURE_AUTO
dmm.digitize.samplerate = 1000000
dmm.digitize.count = 10
print(dmm.digitize.read())
```

Set the digitize function to measure current. Set the sample rate to 1,000,000, with a count of 10, and automatic aperture.

Make a digitize measurement.

Example output:

-0.0039799990218

Example 2

```
channel.setdmm("121:122", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_CURRENT)
channel.setdmm("121:122", dmm.ATTR_DIGI_SAMPLE_RATE, 200000,
    dmm.ATTR_DIGI_APERTURE, dmm.APERTURE_AUTO, dmm.ATTR_DIGI_COUNT, 1)
channel.close("121")
print(dmm.digitize.read())
channel.close("122")
print(dmm.digitize.read())
```

With a 7700 switching module installed, set the measurement function on channels 21 and 22 of slot 1 to digitize current, with a rate of 200,000, automatic aperture, and a count of 1.

Make digitize measurements on each of the channels.

Also see

[Digitize functions](#) (on page 4-35)

[dmm.digitize.aperture](#) (on page 14-148)

[dmm.digitize.samplerate](#) (on page 14-187)

dmm.digitize.dbreference

This attribute defines the decibel (dB) reference setting for the DMM in volts.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1 (1 V)

Usage

```
value = dmm.digitize.dbreference
dmm.digitize.dbreference = value
value = dmm.measure.getattribute(dmm.FUNC_DIGITIZE_VOLTAGE,
    dmm.ATTR_DIGI_DB_REFERENCE)
dmm.measure.setattribute(dmm.FUNC_DIGITIZE_VOLTAGE, dmm.ATTR_DIGI_DB_REFERENCE,
    value)
channel.setdmm("channelList", dmm.ATTR_DIGI_DB_REFERENCE, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_DB_REFERENCE)
```

value	1e-7 V to 1000 V
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

This attribute is only available for the digitize voltage function.

This value only applies when the unit setting for the function is set to decibels.

Example 1

dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE dmm.digitize.unit = dmm.UNIT_DB dmm.digitize.dbreference = 5	Sets the units to decibel and sets the dB reference to 5 for DC volts.

Example 2

channel.setdmm("101:103", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE) channel.setdmm("101:103", dmm.ATTR_DIGI_UNIT, dmm.UNIT_DB, dmm.ATTR_DIGI_DB_REFERENCE, 1e-6, dmm.ATTR_DIGI_SAMPLE_RATE, 1000, dmm.ATTR_DIGI_COUNT, 1000)	Set the measurement function on channels 1 to 3 of slot 1 to digitize voltage, with units set to dB, dB reference of 1e-6 V, sample rate of 1000, and a count of 1000.

Also see

[dmm.digitize.unit](#) (on page 14-189)
[dmm.measure.dbreference](#) (on page 14-223)
[dmm.measure.dbmreference](#) (on page 14-153)

dmm.digitize.dbmreference

This attribute defines the decibel-milliwatts (dBm) reference.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	75 (75 Ω)

Usage

```
value = dmm.digitize.dbmreference
dmm.digitize.dbmreference = value
value = dmm.measure.getattribute(dmm.FUNC_DIGITIZE_VOLTAGE,
    dmm.ATTR_DIGI_DBM_REFERENCE)
dmm.measure.setattribute(dmm.FUNC_DIGITIZE_VOLTAGE, dmm.ATTR_DIGI_DBM_REFERENCE,
    value)
channel.setdmm("channelList", dmm.ATTR_DIGI_DBM_REFERENCE, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_DBM_REFERENCE)
```

value	The dBm reference value (1 Ω to 9999 Ω)
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

This attribute is only available for the digitize voltage function.

This value only applied when the unit setting for the function is set to dBm.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.unit = dmm.UNIT_DBM
dmm.digitize.dbmreference = 85
```

Sets the units to dBm and sets the dBm resistance to 85 Ω.

Example 2

```
channel.setdmm("201:210", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("201:210", dmm.ATTR_DIGI_UNIT, dmm.UNIT_DBM)
channel.setdmm("201:210", dmm.ATTR_DIGI_DBM_REFERENCE, 85)
```

For channels 1 through 10 on slot 2, set the DMM function to digitize voltage.
Set the units to decibel-milliwatts.
Set the dBm reference to 85 Ω.

Also see

- [channel.getdmm](#) (on page 14-75)
- [channel.setdmm](#) (on page 14-88)
- [Show voltage readings in decibel-milliwatts](#) (on page 4-8)
- [dmm.digitize.dbreference](#) (on page 14-152)
- [dmm.digitize.unit](#) (on page 14-189)
- [dmm.measure.dbmreference](#) (on page 14-153)

dmm.digitize.displaydigits

This attribute describes the number of digits that are displayed on the front panel for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.DIGITS_4_5

Usage

```

value = dmm.digitize.displaydigits
dmm.digitize.displaydigits = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_DIGITS)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_DIGITS, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_DIGITS, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_DIGITS)

```

<i>value</i>	6½ display digits: dmm.DIGITS_6_5 5½ display digits: dmm.DIGITS_5_5 4½ display digits: dmm.DIGITS_4_5 3½ display digits: dmm.DIGITS_3_5
<i>function</i>	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

This command affects how the reading for a measurement is displayed on the front panel of the instrument. It does not affect the number of digits returned in a remote command reading. It also does not affect the accuracy or speed of measurements.

The display digits setting is saved with the function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

The change in digits occurs the next time a measurement is made.

To change the number of digits returned in a remote command reading, use `format.asciiprecision`.

Example 1

```

dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.displaydigits = dmm.DIGITS_3_5

```

Set the instrument to use the digitize current measure function.
Set the front panel to display 3½ digits.

Example 2

```

channel.setdmm("101:102", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:102", dmm.ATTR_DIGI_DIGITS, dmm.DIGITS_3_5)

```

Set the measurement function on channels 1 and 2 of slot 1 to digitize voltage. Set the display digits to 3½.

Also see

[format.asciiprecision](#) (on page 14-341)

dmm.digitize.func

This attribute determines which digitize function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.FUNC_NONE

Usage

```
value = dmm.digitize.func
dmm.digitize.func = value
channel.setdmm("channelList", dmm.ATTR_DIGI_FUNCTION, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_FUNCTION)
```

value	The digitize measurement function to make active: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE - No digitize function selected (read only): dmm.FUNC_NONE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

Set this command to the type of measurement you want to digitize.

Reading this command returns the digitize function that is presently active.

If a basic (nondigitize) measurement function is selected, this returns dmm.FUNC_NONE. The none setting is automatically made if you select a function with dmm.measure.func or through the options from the front-panel Measure Functions tab.

If a channel is closed when you assign a function to the channel, all other channels are opened.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
```

Set the measurement function to digitize current.

Example 2

```
channel.setdmm("101:103", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:103", dmm.ATTR_DIGI_RANGE, 100, dmm.ATTR_DIGI_DIGITS,
dmm.DIGITS_5_5, dmm.ATTR_DIGI_SAMPLE_RATE, 1000, dmm.ATTR_DIGI_COUNT, 1000)
```

Set the measurement function on channels 1 to 3 of slot 1 to digitize voltage, with a range of 100, 5½ displayed digits, a sample rate of 1000, and a count of 1000.

Also see

[Digitize functions](#) (on page 4-35)

[dmm.measure.func](#) (on page 14-240)

dmm.digitize.inputimpedance

This attribute determines when the 10 M Ω input divider is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.IMPEDANCE_10M

Usage

```
setting = dmm.digitize.inputimpedance
dmm.digitize.inputimpedance = setting
setting = dmm.measure.getattribute(dmm.FUNC_DIGITIZE_VOLTAGE,
    dmm.ATTR_DIGI_INPUT_IMPEDANCE)
dmm.measure.setattribute(dmm.FUNC_DIGITIZE_VOLTAGE, dmm.ATTR_DIGI_INPUT_IMPEDANCE,
    setting)
channel.setdmm("channelList", dmm.ATTR_DIGI_INPUT_IMPEDANCE, setting)
setting = channel.getdmm("channelList", dmm.ATTR_DIGI_INPUT_IMPEDANCE)
```

<i>setting</i>	10 M Ω for all ranges: dmm.IMPEDANCE_10M Automatic: dmm.IMPEDANCE_AUTO
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

Automatic input impedance provides the lowest measure noise with the highest isolation on the device under test (DUT). When automatic input impedance is selected, the 100 mV to 10 V voltage ranges have more than 10 G Ω input impedance. For the 100 V and 1000 V ranges, a 10 M Ω input divider is placed across the HI and LO input terminals.

When the input impedance is set to 10 M Ω , the 100 mV to 1000 V ranges have a 10 M Ω input divider across the HI and LO input terminals. The 10 M Ω impedance provides stable measurements when the terminals are open (approximately 100 μ V at 1 PLC).

Choosing automatic input impedance is a balance between achieving low DC voltage noise on the 100 mV and 1 V ranges and optimizing measurement noise due to charge injection. The DAQ6510 is optimized for low noise and charge injection when the DUT has less than 100 k Ω input resistance. When the DUT input impedance is more than 100 k Ω , selecting an input impedance of 10 M Ω optimizes the measurement for lowest noise on the 100 mV and 1 V ranges. You can achieve short-term low noise and low charge injection on the 100 mV and 1 V ranges with autozero off. For the 10 V to 1000 V ranges, both input impedance settings achieve low charge injection.

The input impedance setting is only available for the digitize voltage function.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.inputimpedance =
    dmm.IMPEDANCE_AUTO
```

Set input impedance to be set automatically when the digitize voltage function is selected.

Example 2

```
channel.setdmm("101:103", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:103", dmm.ATTR_DIGI_INPUT_IMPEDANCE, dmm.IMPEDANCE_10M)
```

Set the measurement function on channels 1 to 3 of slot 1 to digitize voltage. Set the input impedance to 10 MΩ.

Also see

[dmm.measure.inputimpedance](#) (on page 14-243)

dmm.digitize.limit[Y].audible

This attribute determines if the instrument beeper sounds when a limit test passes or fails.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.AUDIBLE_NONE

Usage

```
value = dmm.digitize.limit[Y].audible
dmm.digitize.limit[Y].audible = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_LIMIT_AUDIBLE_Y)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_LIMIT_AUDIBLE_Y, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_LIMIT_AUDIBLE_Y, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_LIMIT_AUDIBLE_Y)
```

value	When the beeper sounds: - Never: dmm.AUDIBLE_NONE - On test failure: dmm.AUDIBLE_FAIL - On test pass: dmm.AUDIBLE_PASS
Y	Limit number: 1 or 2
function	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

The tone and length of beeper cannot be adjusted.

Example

See [dmm.digitize.limit\[Y\].low.value](#) (on page 14-165) for an example of how to use this command.

Also see

[dmm.digitize.limit\[Y\].enable](#) (on page 14-162)

dmm.digitize.limit[Y].autoclear

This attribute indicates if the test result for limit *Y* should be cleared automatically or not.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.ON

Usage

```
state = dmm.digitize.limit[Y].autoclear
dmm.digitize.limit[Y].autoclear = state
state = dmm.measure.getattribute(function, dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_Y)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_Y, state)
channel.setdmm("channelList", dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_Y, state)
state = channel.getdmm("channelList", dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_Y)
```

<i>state</i>	The auto clear setting: - Disable: dmm.OFF - Enable: dmm.ON
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

When autoclear is set to on, limit conditions are cleared automatically after each measurement. If you are making a series of measurements, the instrument shows the limit test result of the last measurement for the pass or fail indication for the limit.

If you want to know if any of a series of measurements failed the limit, set the autoclear setting to off. When this is set to off, a failed indication is not cleared automatically. It remains set until it is cleared with the clear command.

The autoclear setting affects both the high and low limits.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.limit[1].autoclear = dmm.OFF
```

Turns off autoclear for limit 1 when measuring digitize voltage.

Example 2

```
channel.setdmm("101:103", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:103", dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_1, dmm.OFF)
```

Set the measurement function on channels 1 to 3 of slot 1 to digitize voltage. Set auto clear on limit 2 off.

Also see

[dmm.digitize.limit\[Y\].enable](#) (on page 14-162)

dmm.digitize.limit[Y].clear()

This attribute clears the results of the limit test defined by *Y*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.digitize.limit[Y].clear()
channel.setdmm("channelList", dmm.ATTR_DIGI_LIMIT_FAIL_Y, dmm.FAIL_NONE)
```

<i>Y</i>	Limit number: 1 or 2

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Use this command to clear the test results of limit Y when the limit auto clear option is turned off. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, turn the auto clear option on.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.limit[1].clear()
```

Set the digitize function to voltage.
Clear the results of limit test 1.

Example 2

```
print(channel.getdmm("101:102", dmm.ATTR_DIGI_LIMIT_FAIL_1))
print(channel.getdmm("101:102", dmm.ATTR_DIGI_LIMIT_FAIL_2))

-- Clear limit 1 conditions
channel.setdmm("101:102", dmm.ATTR_DIGI_LIMIT_FAIL_1, dmm.FAIL_NONE)
-- Clear limit 2 conditions
channel.setdmm("101:102", dmm.ATTR_DIGI_LIMIT_FAIL_2, dmm.FAIL_NONE)

print(channel.getdmm("101:102", dmm.ATTR_DIGI_LIMIT_FAIL_1))
print(channel.getdmm("101:102", dmm.ATTR_DIGI_LIMIT_FAIL_2))
```

This example outputs the fail conditions for channels 101 and 102 for limits 1 and 2. It then clears the fail conditions.

Example output showing readings on channels 101 and 102 failed limit 1 low values:

```
[1]=dmm.FAIL_LOW, [2]=dmm.FAIL_LOW
[1]=dmm.FAIL_NONE, [2]=dmm.FAIL_NONE
```

Example output showing the failed conditions are cleared:

```
[1]=dmm.FAIL_NONE, [2]=dmm.FAIL_NONE
[1]=dmm.FAIL_NONE, [2]=dmm.FAIL_NONE
```

Also see

[Digitize functions](#) (on page 4-35)

[dmm.digitize.limit\[Y\].autoclear](#) (on page 14-159)

[dmm.digitize.func](#) (on page 14-155)

dmm.digitize.limit[Y].enable

This attribute enables or disables a limit test on the measurement from the selected digitize function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OFF

Usage

```
state = dmm.digitize.limit[Y].enable
dmm.digitize.limit[Y].enable = state
state = dmm.measure.getattribute(function, dmm.ATTR_DIGI_LIMIT_ENABLE_Y)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_LIMIT_ENABLE_Y, state)
channel.setdmm("channelList", dmm.ATTR_DIGI_LIMIT_ENABLE_Y, state)
state = channel.getdmm("channelList", dmm.ATTR_DIGI_LIMIT_ENABLE_Y)
```

state	Limit Y testing: - Disable: dmm.OFF - Enable: dmm.ON
Y	Limit number: 1 or 2
function	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

This command enables or disables a limit test for the selected digitize function. When this attribute is enabled, the limit Y testing occurs on each measurement made by the instrument. Limit Y testing compares the measurements to the high and low limit values. If a measurement falls outside these limits, the test fails.

Example

See [dmm.digitize.limit\[Y\].low.value](#) (on page 14-165) for examples of how to use this command.

Also see

- [Calculations that you can apply to measurements](#) (on page 4-59)
- [dmm.digitize.limit\[Y\].low.value](#) (on page 14-165)
- [dmm.digitize.limit\[Y\].high.value](#) (on page 14-164)

dmm.digitize.limit[Y].fail

This attribute queries the results of a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = dmm.digitize.limit[Y].fail
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_LIMIT_FAIL_Y)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_LIMIT_FAIL_Y)
```

value	The results of the limit test for limit Y: ■ dmm.FAIL_NONE: Test passed; measurement under or equal to the high limit ■ dmm.FAIL_HIGH: Test failed; measurement exceeded high limit ■ dmm.FAIL_LOW: Test failed; measurement exceeded low limit ■ dmm.FAIL_BOTH: Test failed; measurement exceeded both limits
Y	Limit number: 1 or 2
function	The digitize function: ■ Current: dmm.FUNC_DIGITIZE_CURRENT ■ Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

This command queries the result of a limit test for the selected digitize function.

The response message indicates if the limit test passed or how it failed (on the high or low limit).

If autoclear is set to off, reading the results of a limit test does not clear the fail indication of the test. To clear a failure, send the clear command. To automatically clear the results, set auto clear on.

If auto clear is set to on and you are making a series of measurements, the last measurement limit determines the fail indication for the limit. If auto clear is turned off, the results return a test fail if any of one of the readings failed.

To use this attribute, you must set the limit state to on.

If the readings are stored in a reading buffer, you can use the `bufferVar.statuses` command to see the results.

Example

See [dmm.digitize.limit\[Y\].low.value](#) (on page 14-165) for examples of how to use this command.

Also see

[dmm.digitize.limit\[Y\].enable](#) (on page 14-162)

dmm.digitize.limit[Y].high.value

This attribute specifies the upper limit for a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Save settings Measure configuration list	1

Usage

```
highLimit = dmm.digitize.limit[Y].high.value
dmm.digitize.limit[Y].high.value = highLimit
highLimit = dmm.measure.getattribute(function, dmm.ATTR_DIGI_LIMIT_HIGH_Y)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_LIMIT_HIGH_Y, highLimit)
channel.setdmm("channelList", dmm.ATTR_DIGI_LIMIT_HIGH_Y, highLimit)
highLimit = channel.getdmm("channelList", dmm.ATTR_DIGI_LIMIT_HIGH_Y)
```

<i>highLimit</i>	The value of the upper limit (-1e+12 to 1e+12)
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

This command sets the high limit for the limit Y test for the selected digitize function. When limit Y testing is enabled, the instrument generates a fail indication when the measurement value is more than this value.

Example

See [dmm.digitize.limit\[Y\].low.value](#) (on page 14-165) for an example of how to use this command.

Also see

- [dmm.digitize.limit\[Y\].enable](#) (on page 14-162)
- [dmm.digitize.limit\[Y\].low.value](#) (on page 14-165)

dmm.digitize.limit[Y].low.value

This attribute specifies the lower limit for a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	-1

Usage

```
lowLimit = dmm.digitize.limit[Y].low.value
dmm.digitize.limit[Y].low.value = lowLimit
lowLimit = dmm.measure.getattribute(function, dmm.ATTR_DIGI_LIMIT_LOW_Y)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_LIMIT_LOW_Y, lowLimit)
channel.setdmm("channelList", dmm.ATTR_DIGI_LIMIT_LOW_Y, lowLimit)
lowLimit = channel.getdmm("channelList", dmm.ATTR_DIGI_LIMIT_LOW_Y)
```

<i>lowLimit</i>	The low limit value of limit <i>Y</i> : -1E+12 to 1E+12)
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The digitize function: ■ Current: dmm.FUNC_DIGITIZE_CURRENT ■ Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

This command sets the lower limit for the limit *Y* test for the selected digitize function. When limit *Y* testing is enabled, this causes a fail indication to occur when the measurement value is less than this value.

Example 1

This example enables limits 1 and 2 for digitize voltage measurements. Limit 1 is checking for readings to be between 3 V and 5 V, while limit 2 is checking for the readings to be between 1 V and 7 V. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail is 1. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
reset()
-- Set the instrument to measure digitized voltage
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
-- Set the range to 10 V
dmm.digitize.range = 10
-- Disable auto clearing for limit 1
dmm.digitize.limit[1].autoclear = dmm.OFF
-- Set high limit on 1 to fail if reading exceeds 5 V
dmm.digitize.limit[1].high.value = 5
-- Set low limit on 1 to fail if reading is less than 3 V
dmm.digitize.limit[1].low.value = 3
-- Enable limit 1 checking for digitized voltage measurements
dmm.digitize.limit[1].enable = dmm.ON
-- Disable auto clearing for limit 2
dmm.digitize.limit[2].autoclear = dmm.OFF
-- Set high limit on 2 to fail if reading exceeds 7 V
dmm.digitize.limit[2].high.value = 7
-- Set low limit on 2 to fail if reading is less than 1 V
dmm.digitize.limit[2].low.value = 1
-- Set the beeper to sound if the reading exceeds the limits for limit 2
dmm.digitize.limit[2].audible = dmm.AUDIBLE_FAIL
-- Enable limit 2 checking for digitized voltage measurements
dmm.digitize.limit[2].enable = dmm.ON
-- Set the digitize count to 50
dmm.digitize.count = 50
-- Create a reading buffer that can store 100 readings
LimitBuffer = buffer.make(100)
-- Make 50 readings and store them in LimitBuffer
dmm.digitize.read(LimitBuffer)
-- Check if any of the 50 readings were outside of the limits
print("limit 1 results = " .. dmm.digitize.limit[1].fail)
print("limit 2 results = " .. dmm.digitize.limit[2].fail)
-- Clear limit 1 conditions
dmm.digitize.limit[1].clear()
-- Clear limit 2 conditions
dmm.digitize.limit[2].clear()
```

Example output that shows all readings are within limit values (all readings between 3 V and 5 V):

```
limit 1 results = dmm.FAIL_NONE
```

```
limit 2 results = dmm.FAIL_NONE
```

Example output showing at least one reading failed limit 1 high values (a 6 V reading would cause this condition or a reading greater than 5 V but less than 7 V):

```
limit 1 results = dmm.FAIL_HIGH
```

```
limit 2 results = dmm.FAIL_NONE
```

Example output showing at least one reading failed limit 1 and 2 low values (a 0.5 V reading would cause this condition or a reading less than 1 V):

```
limit 1 results = dmm.FAIL_LOW
```

```
limit 2 results = dmm.FAIL_LOW
```

Example 2

This example enables limits 1 and 2 for digitize voltage measurements on channels 1 and 2 of slot 1. Limit 1 is checking for readings to be between 3 and 5 V, while limit 2 is checking for the readings to be between 1 and 7 V. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail is 1. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.


```
reset()
sampleCount = 50
scanCount = 25

-- Set channels 1 and 2 to measure digitized voltage.
-- Each has a sample count of 50.
channel.setdmm("101:102", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE,
    dmm.ATTR_DIGI_COUNT, sampleCount)

-- Set the range to 10 V.
channel.setdmm("101:102", dmm.ATTR_DIGI_RANGE, 10)

-- Set up limit 1: Disable auto clearing, set the high limit
-- to fail if reading exceeds 5 V, set low limit to fail if
-- reading is less than 3 V, and enable limit checking.
channel.setdmm("101:102", dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_1, dmm.OFF,
    dmm.ATTR_DIGI_LIMIT_HIGH_1, 5, dmm.ATTR_DIGI_LIMIT_LOW_1, 3,
    dmm.ATTR_DIGI_LIMIT_ENABLE_1, dmm.ON)

-- Set up limit 2: Disable auto clearing, set the high limit
-- to fail if reading exceeds 7 V, set low limit to fail if
-- reading is less than 1 V, and enable limit checking.
channel.setdmm("101:102", dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_2, dmm.OFF,
    dmm.ATTR_DIGI_LIMIT_HIGH_2, 7, dmm.ATTR_DIGI_LIMIT_LOW_2, 1,
    dmm.ATTR_DIGI_LIMIT_ENABLE_2, dmm.ON)

--- Set the beeper to sound if the reading exceeds the limits for limit 2.
channel.setdmm("101:102", dmm.ATTR_DIGI_LIMIT_AUDIBLE_2, dmm.AUDIBLE_FAIL)

-- Set the digitize count to 50.
-- channel.setdmm("101:102", dmm.ATTR_DIGI_COUNT, 50)

-- Create a standard reading buffer that can store 100 readings.
LimitBuffer = buffer.make(2 * sampleCount * scanCount, buffer.STYLE_STANDARD)

-- Make 2500 readings and store them in LimitBuffer.
scan.create("101:102")
scan.buffer = LimitBuffer
LimitBuffer.clear()
scan.scancount = scanCount
scan.scaninterval = 1.0

trigger.model.initiate()
waitcomplete()

-- Check if any of the 50 readings were outside of the limits.
print("limit 1 results = " .. dmm.digitize.limit[1].fail)
print("limit 2 results = " .. dmm.digitize.limit[2].fail)
-- Clear limit 1 conditions
dmm.digitize.limit[1].clear()
-- clear limit 2 conditions
dmm.digitize.limit[2].clear()

printbuffer(1, LimitBuffer.n, LimitBuffer)
```

Example output that shows all readings are within limit values (all readings between 3 V and 5 V):

```
limit 1 results = dmm.FAIL_NONE
limit 2 results = dmm.FAIL_NONE
```

Example output showing at least one reading failed limit 1 high values (a 6 V reading would cause this condition or a reading greater than 5 V but less than 7 V):

```
limit 1 results = dmm.FAIL_HIGH
limit 2 results = dmm.FAIL_NONE
```

Example output showing at least one reading failed limit 1 and 2 low values (a 0.5 V reading would cause this condition or a reading less than 1 V):

```
limit 1 results = dmm.FAIL_LOW
limit 2 results = dmm.FAIL_LOW
```

Also see

- [dmm.digitize.limit\[Y\].autoclear](#) (on page 14-159)
- [dmm.digitize.limit\[Y\].clear\(\)](#) (on page 14-160)
- [dmm.digitize.limit\[Y\].enable](#) (on page 14-162)
- [dmm.digitize.limit\[Y\].fail](#) (on page 14-163)
- [dmm.digitize.limit\[Y\].high.value](#) (on page 14-164)

dmm.digitize.math.enable

This attribute enables or disables math operations on measurements for the selected digitize function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OFF

Usage

```
value = dmm.digitize.math.enable
dmm.digitize.math.enable = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_MATH_ENABLE)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_MATH_ENABLE, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_MATH_ENABLE, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_MATH_ENABLE)
```

value	The math enable setting: - Disable: dmm.OFF - Enable: dmm.ON
function	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

When this command is set to on, the math operation specified by the math format command is performed before completing a measurement.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.math.format = dmm.MATH_PERCENT
dmm.digitize.count = 1
dmm.digitize.math.percent = dmm.digitize.read()
dmm.digitize.math.enable = dmm.ON
dmm.digitize.count = 5
MathBuffer = buffer.make(100)
dmm.digitize.read(MathBuffer)
printbuffer(1, MathBuffer.n, MathBuffer.formattedreadings)
dmm.digitize.count = 1
for x = 1, 3 do
    print(dmm.digitize.read(MathBuffer))
end
```

Configure the instrument for digitize voltage.

Set math format to percent.

Acquire 1 reading to use as the relative percent value.

Take 5 readings with percent math enabled and store them in a buffer called `MathBuffer` that can store 100 readings.

Take three additional readings.

Sample output assuming no load was connected to the instrument:

```
-100.00 %, -100.00 %, -100.00 %, -100.00 %, -100.00 %
-100.00058257
-99.999126228
-99.998932056
```

Example 2

```
-- Set channel 1 on slot 1 to use the digitize voltage function.
channel.setdmm("101", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
-- Set channel 1 to use the percentage math format and enable math.
channel.setdmm("101", dmm.ATTR_DIGI_MATH_FORMAT, dmm.MATH_PERCENT,
    dmm.ATTR_DIGI_MATH_ENABLE, dmm.ON )

-- Acquire one reading to use as the relative percent value.
channel.close("101")
dmm.digitize.count = 1
dmm.digitize.math.percent = dmm.digitize.read()
channel.setdmm("101", dmm.ATTR_DIGI_MATH_PERCENT, dmm.digitize.read())

-- Create a buffer named MathBuffer that holds 100 readings.
MathBuffer = buffer.make(100)
-- Set the measure count to 100
channel.setdmm("101", dmm.ATTR_DIGI_COUNT, 100)

dmm.digitize.read(MathBuffer)

printbuffer(1, MathBuffer.n, MathBuffer.formattedreadings)
```

Set the instrument to digitize voltage.
Set math format to percent for channel 1 on slot 1.
Acquire 1 reading to use as the relative percent value.
Close channel 1.
Set channel 1 to use the math format percent.
Create a buffer.
Set the measure count for channel 1 to 100.
Make readings.
Output the data.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.format](#) (on page 14-172)

dmm.digitize.math.format

This attribute specifies which math operation is performed on measurements when math operations are enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.MATH_PERCENT

Usage

```
operation = dmm.digitize.math.format
dmm.digitize.math.format = operation
operation = dmm.measure.getattribute(function, dmm.ATTR_DIGI_MATH_FORMAT)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_MATH_FORMAT, operation)
channel.setdmm("channelList", dmm.ATTR_DIGI_MATH_FORMAT, operation)
operation = channel.getdmm("channelList", dmm.ATTR_DIGI_MATH_FORMAT)
```

<i>operation</i>	Math operation to be performed on measurements: ■ y = mx+b: dmm.MATH_MXB ■ Percent: dmm.MATH_PERCENT ■ Reciprocal: dmm.MATH_RECIPROCAL
<i>function</i>	The digitize function: ■ Current: dmm.FUNC_DIGITIZE_CURRENT ■ Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

This specifies which math operation is performed on measurements for the selected digitize function. You can choose one of the following math operations:

- **y = mx+b**: Manipulate normal display readings by adjusting the m and b factors.
- **Percent**: Displays measurements as the percentage of deviation from a specified reference constant.
- **Reciprocal**: The reciprocal math operation displays measurement values as reciprocals. The displayed value is $1/X$, where X is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).

Math calculations are applied to the input signal after relative offset and before limit tests.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.math.format = dmm.MATH_RECIPROCAL
dmm.digitize.math.enable = dmm.ON
```

Enables the reciprocal math operation on digitize voltage measurements.

Example 2

```
-- Set channel 1 on slot 1 to use the digitize voltage function.  
channel.setdmm("101", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)  
-- Set channel 1 to use the reciprocal math format and enable math.  
channel.setdmm("101", dmm.ATTR_DIGI_MATH_FORMAT, dmm.MATH_RECIPROCAL,  
    dmm.ATTR_DIGI_MATH_ENABLE, dmm.ON)
```

Enables the reciprocal math operation on digitize voltage measurements made on channel 1 of slot 1.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.enable](#) (on page 14-169)

dmm.digitize.math.mxb.bfactor

This attribute specifies the offset, b, for the $y = mx + b$ operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0

Usage

```
offsetFactor = dmm.digitize.math.mxb.bfactor
dmm.digitize.math.mxb.bfactor = offsetFactor
offsetFactor = dmm.measure.getattribute(function, dmm.ATTR_DIGI_MATH_MXB_BF)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_MATH_MXB_BF, offsetFactor)
channel.setdmm("channelList", dmm.ATTR_DIGI_MATH_MXB_BF, offsetFactor)
offsetFactor = channel.getdmm("channelList", dmm.ATTR_DIGI_MATH_MXB_BF)
```

<i>offsetFactor</i>	The offset for the $y = mx + b$ operation: $-1e12$ to $+1e12$
<i>function</i>	The digitize function: - Current: <code>dmm.FUNC_DIGITIZE_CURRENT</code> - Voltage: <code>dmm.FUNC_DIGITIZE_VOLTAGE</code>
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

This attribute specifies the offset (b) for an $mx+b$ operation.

The $mx+b$ math operation lets you manipulate normal display readings (x) mathematically based on the calculation:

$$y = mx + b$$

Where:

- y is the displayed result
- m is a user-defined constant for the scale factor
- x is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- b is the user-defined constant for the offset factor

Example 1

<pre>dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT dmm.digitize.math.format = dmm.MATH_MXB dmm.digitize.math.mxb.mfactor = 0.80 dmm.digitize.math.mxb.bfactor = 42 dmm.digitize.math.enable = dmm.ON</pre>	Set the digitize function to digitize current. Set the scale factor for the $mx + b$ operation to 0.80. Set the offset factor to 42. Enable the math function.

Example 2

```
-- Set channel 1 on slot 1 to use the digitize voltage function.  
channel.setdmm("101", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)  
-- Set channel 1 to use the MXB math format, set up the factors, and enable math.  
channel.setdmm("101", dmm.ATTR_DIGI_MATH_FORMAT, dmm.MATH_MXB,  
    dmm.ATTR_DIGI_MATH_MXB_MF, 0.8, dmm.ATTR_DIGI_MATH_MXB_BF, 42,  
    dmm.ATTR_DIGI_MATH_ENABLE, dmm.ON)
```

Enables the $y = mx + b$ math operation on digitize voltage measurements made on channel 1 of slot 1. Set the m (scale) factor to 0.8 and the b (offset) factor to 42.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.enable](#) (on page 14-169)

[dmm.digitize.math.format](#) (on page 14-172)

[dmm.digitize.math.mxb.mfactor](#) (on page 14-176)

dmm.digitize.math.mxb.mfactor

This attribute specifies the scale factor, m , for the $y = mx + b$ math operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1

Usage

```
scaleFactor = dmm.digitize.math.mxb.mfactor
dmm.digitize.math.mxb.mfactor = scaleFactor
scaleFactor = dmm.measure.getattribute(function, dmm.ATTR_DIGI_MATH_MXB_MF)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_MATH_MXB_MF, scaleFactor)
channel.setdmm("channelList", dmm.ATTR_DIGI_MATH_MXB_MF, scaleFactor)
scaleFactor = channel.getdmm("channelList", dmm.ATTR_DIGI_MATH_MXB_MF)
```

<i>scaleFactor</i>	The scale factor: $-1e12$ to $+1e12$
<i>function</i>	The digitize function: - Current: <code>dmm.FUNC_DIGITIZE_CURRENT</code> - Voltage: <code>dmm.FUNC_DIGITIZE_VOLTAGE</code>
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

This command sets the scale factor (m) for an $mx + b$ operation for the selected measurement function.

The $mx + b$ math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$$y = mx + b$$

Where:

- y is the displayed result
- m is a user-defined constant for the scale factor
- x is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- b is the user-defined constant for the offset factor

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.math.format = dmm.MATH_MXB
dmm.digitize.math.mxb.mfactor = 0.80
dmm.digitize.math.mxb.bfactor = 42
dmm.digitize.math.enable = dmm.ON
```

Set the digitize function to digitize current.
Set the scale factor for the $mx + b$ operation to 0.80.
Set the offset factor to 42.
Enable the math function.

Example 2

```
-- Set channel 1 on slot 1 to use the digitize voltage function.
channel.setdmm("101", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
-- Set channel 1 to use the MXB math format, set up the factors, and enable math.
channel.setdmm("101", dmm.ATTR_DIGI_MATH_FORMAT, dmm.MATH_MXB,
    dmm.ATTR_DIGI_MATH_MXB_MF, 0.8, dmm.ATTR_DIGI_MATH_MXB_BF, 42,
    dmm.ATTR_DIGI_MATH_ENABLE, dmm.ON)
```

Enables the $y = mx + b$ math operation on digitize voltage measurements made on channel 1 of slot 1. Set the m (scale) factor to 0.8 and the b (offset) factor to 42.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.enable](#) (on page 14-169)

[dmm.digitize.math.format](#) (on page 14-172)

[dmm.digitize.math.mxb.bfactor](#) (on page 14-174)

dmm.digitize.math.percent

This attribute specifies the reference constant that is used when math operations are set to percent.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1

Usage

```
value = dmm.digitize.math.percent
dmm.digitize.math.percent = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_MATH_PERCENT)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_MATH_PERCENT, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_MATH_PERCENT, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_MATH_PERCENT)
```

<i>value</i>	The reference used when the math operation is set to percent: $-1e12$ to $+1e12$
<i>function</i>	The digitize function: - Current: <code>dmm.FUNC_DIGITIZE_CURRENT</code> - Voltage: <code>dmm.FUNC_DIGITIZE_VOLTAGE</code>
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

The percent math function displays measurements as percent deviation from a specified reference constant. The percent calculation is:

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

Where:

- *Percent* is the result
- *Input* is the measurement (if relative offset is being used, this is the relative offset value)
- *Reference* is the user-specified constant

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.math.format = dmm.MATH_PERCENT
dmm.digitize.math.percent = 42
dmm.digitize.math.enable = dmm.ON
```

Set the measurement function to digitize current.
Set the math operations to percent.
Set the percentage value to 42.
Enable math operations.

Example 2

```
-- Set channel 1 on slot 1 to use the digitize voltage function.
channel.setdmm("101", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
-- Set channel 1 to use the percentage math format, set the percentage,
-- and enable math.
channel.setdmm("101", dmm.ATTR_DIGI_MATH_FORMAT, dmm.MATH_PERCENT,
    dmm.ATTR_DIGI_MATH_PERCENT, 42, dmm.ATTR_DIGI_MATH_ENABLE, dmm.ON)
```

Enables the $y = mx + b$ math operation on digitize voltage measurements made on channel 1 of slot 1. Set the m (scale) factor to 0.8 and the b (offset) factor to 42.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.enable](#) (on page 14-169)

[dmm.digitize.math.format](#) (on page 14-172)

dmm.digitize.range

This attribute determines the positive full-scale measure range for the selected digitize function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Current: 1 A Voltage: 10 V

Usage

```
value = dmm.digitize.range
dmm.digitize.range = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_RANGE)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_RANGE, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_RANGE, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_RANGE)
```

<i>value</i>	Set to the maximum expected value to be measured: - Current: 100 μA to 3 A - Voltage: 100 mV to 1000 V
<i>function</i>	The digitize function to which to assign this parameter: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

When you assign a range value, the instrument selects a fixed range that is large enough to measure the assigned value. The instrument selects the best range for measuring the maximum expected value.

For example, for digitize current measurements, if you expect a reading of approximately 9 mA, set the range to 9 mA to select the 10 mA range.

When you read this setting, you see the positive full-scale value of the measurement range that the instrument is presently using.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.range = 90
```

Set the function to digitize voltage. Set the range to 90 V, which selects the 100 V range.

Example 2

```
channel.setdmm("101", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101", dmm.ATTR_DIGI_RANGE, 90)
```

Set channel 1 on slot 1 to the digitize voltage function. Set the range of channel 1 to 90 V, which selects the 100 V range.

Also see

None

dmm.digitize.read()

This function makes digitize measurements, places them in a reading buffer, and returns the last reading.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = dmm.digitize.read()  
reading = dmm.digitize.read(bufferName)
```

<i>reading</i>	The last reading of the measurement process
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (<i>defbuffer1</i> or <i>defbuffer2</i>) or a user-defined buffer; if nothing is specified, the reading is stored in <i>defbuffer1</i>

Details

You must set the instrument to make digitize measurements before sending this command with the `dmm.digitize.func` attribute.

This command initiates measurements using the present function settings, stores the readings in a reading buffer, and returns the last reading.

This command makes the number of digitize measurements that is set by the `dmm.digitize.count` attribute.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

Example 1

<pre>voltMeasBuffer = buffer.make(10000) dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE print(dmm.digitize.read(voltMeasBuffer))</pre>	Create a buffer named <code>voltMeasBuffer</code> . Set the instrument to digitize voltage. Make a measurement that is stored in the <code>voltMeasBuffer</code> and is also printed.

Example 2

```
reset()
channel.setdmm("101", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.close("101")
print(dmm.digitize.read())
```

Set up channel 1 to digitize voltage.
Close channel 1.
Digitize on the closed channel.

Also see

[buffer.make\(\)](#) (on page 14-22)
[dmm.digitize.count](#) (on page 14-150)
[dmm.digitize.func](#) (on page 14-155)
[dmm.digitize.unit](#) (on page 14-189)
[Reading buffers](#) (on page 6-1, on page 6-1)
[trigger.model.load\(\) — SimpleLoop](#) (on page 14-487)

dmm.digitize.readwithtime()

This function initiates digitize measurements and returns the last actual measurement and time information in UTC format without using the trigger mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading, seconds, fractional = dmm.digitize.readwithtime()
reading, seconds, fractional = dmm.digitize.readwithtime(bufferName)
```

<i>reading</i>	The last reading of the measurement process
<i>seconds</i>	Seconds in UTC format
<i>fractional</i>	Fractional seconds
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; if no buffer is specified, this parameter defaults to defbuffer1

Details

This command initiates digitize measurements using the present function settings, stores the readings in a reading buffer, and returns the last reading.

The `dmm.digitize.count` attribute determines how many measurements are performed.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

Example 1

```
print(dmm.digitize.readwithtime(defbuffer1))
```

Print the last digitize measurement and time information from `defbuffer1` in UTC format, which will look similar to:

```
-0.0003882925875    1415795836    0.946164546
```

Example 2

```
channel.close("101")  
print(dmm.digitize.readwithtime())  
channel.close("109")  
print(dmm.digitize.readwithtime())
```

Close channel 1 of slot 1.
Print the digitize reading.
Close channel 9 of slot 1.
Print the digitize reading.

Also see

[dmm.digitize.count](#) (on page 14-150)

[dmm.digitize.func](#) (on page 14-155)

[trigger.model.load\(\) — SimpleLoop](#) (on page 14-487)

dmm.digitize.rel.acquire()

This function acquires a measurement and stores it as the relative offset value for the selected digitize function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

dmm.digitize.rel.acquire()

Details

This command triggers the instrument to make a new measurement for the selected function. This measurement is then stored as the new relative offset level.

When you send this command, the instrument does not apply any math, limit test, or filter settings to the measurement, even if they are set. It is a measurement that is made as if these settings are disabled.

If an error event occurs during the measurement, `nil` is returned and the relative offset level remains at the last valid setting.

You must change to the function for which you want to acquire a value before sending this command.

The instrument must have relative offset enabled to use the acquired relative offset value.

After executing this command, you can use the `dmm.digitize.rel.level` attribute to see the last relative level value that was acquired or that was set.

Example

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
rel_value = dmm.digitize.rel.acquire()
dmm.digitize.rel.enable = dmm.ON
print(rel_value)
```

Acquires a relative offset level value for the digitize current function and turns the relative offset feature on. Output the value of the offset.

Also see

- [dmm.digitize.func](#) (on page 14-155)
- [dmm.digitize.rel.enable](#) (on page 14-184)
- [dmm.digitize.rel.level](#) (on page 14-185)

dmm.digitize.rel.enable

This attribute enables or disables the application of a relative offset value to the measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OFF

Usage

```
state = dmm.digitize.rel.enable
dmm.digitize.rel.enable = state
state = dmm.measure.getattribute(function, dmm.ATTR_DIGI_REL_ENABLE)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_REL_ENABLE, state)
channel.setdmm("channelList", dmm.ATTR_DIGI_REL_ENABLE, state)
state = channel.getdmm("channelList", dmm.ATTR_DIGI_REL_ENABLE)
```

state	The setting: ■ Enable: dmm.ON ■ Disable: dmm.OFF
function	The digitize function: ■ Current: dmm.FUNC_DIGITIZE_CURRENT ■ Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

When relative measurements are enabled, all subsequent digitized readings are offset by the relative offset value that was calculated when you acquired the relative offset value.

Each returned measured relative reading is the result of the following calculation:

$$\text{Displayed reading} = \text{Actual measured reading} - \text{Relative offset value}$$

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.rel.acquire()
dmm.digitize.rel.enable = dmm.ON
```

Enables the relative measurements for digitize current after using the acquire command to set the relative level.

Example 2

```
channel.setdmm("101:103", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:103", dmm.ATTR_DIGI_REL_LEVEL, 1, dmm.ATTR_DIGI_REL_ENABLE,
dmm.ON)
```

Set up channels 1 to 3 on slot 1 for the digitize voltage function. Set the relative level to 1 V and apply the relative offset.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.rel.acquire\(\)](#) (on page 14-183)

[dmm.digitize.rel.level](#) (on page 14-185)

dmm.digitize.rel.level

This attribute contains the relative offset value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0

Usage

```
value = dmm.digitize.rel.level
dmm.digitize.rel.level = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_REL_LEVEL)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_REL_LEVEL, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_REL_LEVEL, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_REL_LEVEL)
```

value	Relative offset value for digitized measurements: - Current: -3 to 3 - Voltage: -1000 to 1000
function	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

This command specifies the relative offset value that can be applied to new digitized measurements. When relative offset is enabled, all subsequent digitized readings are offset by the value that is set for this command.

You can set this value, or have the instrument acquire a value. If the instrument acquires the value, read this setting to return the value that was measured internally.

NOTE

If you have math, limits, or filter operations selected, you can set the relative offset value to include the adjustments made by these operations. To include these operations, set `dmm.digitize.rel.level` to `dmm.digitize.read()`. The adjustments from these operations are not used if you use the `dmm.digitize.rel.acquire()` function to set the relative offset level.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.rel.level = dmm.digitize.read()
dmm.digitize.rel.enable = dmm.ON
```

Set the digitize function to digitize current.
Set the relative offset level to be the reading with any calculations included.
Enable the relative offset.

Example 2

```
channel.setdmm("101:103", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:103", dmm.ATTR_DIGI_REL_LEVEL, 1, dmm.ATTR_DIGI_REL_ENABLE,
dmm.ON)
```

Set up channels 1 to 3 on slot 1 for the digitize voltage function. Set the relative level to 1 V and apply the relative offset.

Also see

[Relative offset](#) (on page 4-56)

[dmm.digitize.rel.acquire\(\)](#) (on page 14-183)

[dmm.digitize.rel.enable](#) (on page 14-184)

dmm.digitize.samplerate

This attribute defines the precise acquisition rate at which the digitizing measurements are made.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1,000,000

Usage

```

readings = dmm.digitize.samplerate
dmm.digitize.samplerate = readings
readings = dmm.measure.getattribute(function, dmm.ATTR_DIGI_SAMPLE_RATE)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_SAMPLE_RATE, readings)
channel.setdmm("channelList", dmm.ATTR_DIGI_SAMPLE_RATE, readings)
readings = channel.getdmm("channelList", dmm.ATTR_DIGI_SAMPLE_RATE)

```

<i>readings</i>	The number of readings per second: 1,000 to 1,000,000
<i>function</i>	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

The sample rate determines how fast the DAQ6510 acquires a digitized reading.

Set the sample rate before setting the aperture. If the aperture setting is too high for the selected sample rate, it is automatically adjusted to the highest aperture that can be used with the sample rate.

Example 1

```

dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.samplerate = 200000
dmm.digitize.aperture = dmm.APERTURE_AUTO
dmm.digitize.count = 1
print(dmm.digitize.read())

```

Set the digitize function to digitize current. Set the sample rate to 200,000, with a count of 1, and automatic aperture.

Make a digitize measurement.

Example 2

```

channel.setdmm("121:122", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_CURRENT)
channel.setdmm("121:122", dmm.ATTR_DIGI_SAMPLE_RATE, 200000,
    dmm.ATTR_DIGI_APERTURE, dmm.APERTURE_AUTO, dmm.ATTR_DIGI_COUNT, 1)
channel.close("121")
print(dmm.digitize.read())
channel.close("122")
print(dmm.digitize.read())

```

With a 7700 switching module installed, set the measurement function on channels 21 and 22 of slot 1 to digitize current, with a rate of 200,000, automatic aperture, and a count of 1.

Make digitize measurements on each of the channels.

Also see

[dmm.digitize.aperture](#) (on page 14-148)

[dmm.digitize.count](#) (on page 14-150)

dmm.digitize.unit

This attribute sets the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.UNIT_VOLT (digitize voltage function) dmm.UNIT_AMP (digitize current function)

Usage

```
value = dmm.digitize.unit
dmm.digitize.unit = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_UNIT)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_UNIT, value)
channel.setdmm("channelList", dmm.ATTR_DIGI_UNIT, value)
value = channel.getdmm("channelList", dmm.ATTR_DIGI_UNIT)
```

value	Units to display for the digitize voltage function: - Volts: dmm.UNIT_VOLT - Decibels: dmm.UNIT_DB - Decibel-milliwatts: dmm.UNIT_DBM Units to display for the digitize current function: - Amps: dmm.UNIT_AMP
function	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
channelList	The channels to set, using standard channel naming (on page 14-3)

Details

The change in measurement units is displayed when the next measurement is made. You can only change the units for the listed functions.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_VOLTAGE
dmm.digitize.unit = dmm.UNIT_DB
```

Set the measure function to digitize voltage.
Set the units to display in decibels.

Example 2

```
channel.setdmm("101:103", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:103", dmm.ATTR_DIGI_UNIT, dmm.UNIT_DB,
    dmm.ATTR_DIGI_DB_REFERENCE, 1e-6, dmm.ATTR_DIGI_SAMPLE_RATE, 1000,
    dmm.ATTR_DIGI_COUNT, 1000)
```

Set the measurement function on channels 1 to 3 of slot 1 to digitize voltage, with units set to dB , dB reference of 1e-6 V, sample rate of 1000, and a count of 1000.

Also see

- [dmm.digitize.dbreference](#) (on page 14-152)
- [dmm.digitize.dbmreference](#) (on page 14-153)
- [dmm.digitize.func](#) (on page 14-155)
- [Show voltage readings in decibels](#) (on page 4-8)
- [Show voltage readings in decibel-milliwatts \(dBm\)](#) (on page 4-8)

dmm.digitize.userdelay[N]

This attribute sets a user-defined delay that you can use in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0 (0 s)

Usage

```
delayTime = dmm.digitize.userdelay[N]
dmm.digitize.userdelay[N] = delayTime
delayTime = dmm.measure.getattribute(function, dmm.ATTR_DIGI_USER_DELAY_N)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_USER_DELAY_N, delayTime)
channel.setdmm("channelList", dmm.ATTR_DIGI_USER_DELAY_N, delayTime)
delayTime = channel.getdmm("channelList", dmm.ATTR_DIGI_USER_DELAY_N)
```

<i>delayTime</i>	The delay: 0 for no delay, or 167 ns to 10 ks
<i>N</i>	The user delay to which this time applies: 1 to 5
<i>function</i>	The digitize function: - Current: dmm.FUNC_DIGITIZE_CURRENT - Voltage: dmm.FUNC_DIGITIZE_VOLTAGE
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Details

To use this command in a trigger model, assign the delay to the dynamic delay block.
The delay is specific to the selected function.

Example 1

```
dmm.digitize.func = dmm.FUNC_DIGITIZE_CURRENT
dmm.digitize.userdelay[2] = 0.5
trigger.model.setblock(6, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M2)
Set user delay 2 to be 0.5 s. Sets trigger-model block 6 to use the delay.
```

Example 2

```
channel.setdmm("101:102", dmm.ATTR_DIGI_FUNCTION, dmm.FUNC_DIGITIZE_VOLTAGE)
channel.setdmm("101:102", dmm.ATTR_DIGI_USER_DELAY_2, 0.5)
trigger.model.setblock(6, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M2)
Set user delay 2 to be 0.5 s for channels 1 to 2 of slot 1. Set trigger-model block 6 to use the delay.
```

Also see

[trigger.model.setblock\(\)](#) — [trigger.BLOCK_DELAY_DYNAMIC](#) (on page 14-508)

dmm.measure.analogtrigger.edge.level

This attribute defines the signal level that generates the analog trigger event for the edge trigger mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0

Usage

```
value = dmm.measure.analogtrigger.edge.level
dmm.measure.analogtrigger.edge.level = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL)
```

<i>value</i>	The signal level that generates the trigger
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command is only available when the analog trigger mode is set to edge.

The edge level can be set to any value in the active measurement range.

To use the analog trigger with the measure functions, a range must be set (you cannot use autorange) and autozero must be disabled.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_CURRENT
dmm.measure.range = 3
dmm.measure.autozero.enable = dmm.OFF
dmm.measure.analogtrigger.mode = dmm.MODE_EDGE
dmm.measure.analogtrigger.edge.level = 2.5
dmm.measure.analogtrigger.edge.slope = dmm.SLOPE_FALLING
```

Set measure function to DC current.

Set range to 3 A.

Disable autozero.

Set the analog trigger mode to edge.

Set the analog trigger level to 2.5 A.

Set the level to be detected on a falling edge.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 3)
channel.setdmm("101:109", dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_MODE, dmm.MODE_EDGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE, dmm.SLOPE_FALLING)
```

For channels 1 through 9 on slot 1, set the measure function to DC voltage.

Set range to 3 V.

Disable autozero.

Set the analog trigger mode to edge.

Set the analog trigger level to 5 V.

Set the level to be detected on a falling edge.

Also see

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.edge.level](#) (on page 14-137)

[dmm.measure.analogtrigger.edge.slope](#) (on page 14-193)

[dmm.measure.analogtrigger.mode](#) (on page 14-195)

dmm.measure.analogtrigger.edge.slope

This attribute defines the slope of the analog trigger edge.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.SLOPE_RISING

Usage

```
value = dmm.measure.analogtrigger.edge.slope
dmm.measure.analogtrigger.edge.slope = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE)
```

<i>value</i>	The slope of the analog trigger edge: ■ Rising: dmm.SLOPE_RISING ■ Falling: dmm.SLOPE_FALLING
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This is only available when the analog trigger mode is set to edge.

Rising causes an analog trigger event when the analog signal trends from below the analog signal level to above the level.

Falling causes an analog trigger event when the signal trends from above to below the level.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_CURRENT
dmm.measure.range = 3
dmm.measure.autozero.enable = dmm.OFF
dmm.measure.analogtrigger.mode = dmm.MODE_EDGE
dmm.measure.analogtrigger.edge.level = 2.5
dmm.measure.analogtrigger.edge.slope = dmm.SLOPE_FALLING
```

Set measure function to DC current.
Set range to 3 A.
Disable autozero.
Set the analog trigger mode to edge.
Set the analog trigger level to 2.5 A.
Set the level to be detected on a falling edge.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 3)
channel.setdmm("101:109", dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_MODE, dmm.MODE_EDGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE, dmm.SLOPE_FALLING)
```

For channels 1 through 9 on slot 1, set the measure function to DC voltage.

Set range to 3 V.

Disable autozero.

Set the analog trigger mode to edge.

Set the analog trigger level to 5 V.

Set the level to be detected on a falling edge.

Also see

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.edge.slope](#) (on page 14-138)

[dmm.measure.analogtrigger.edge.level](#) (on page 14-191)

[dmm.measure.analogtrigger.mode](#) (on page 14-195)

dmm.measure.analogtrigger.mode

This attribute configures the type of signal behavior that can generate an analog trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.MODE_OFF

Usage

```

setting = dmm.measure.analogtrigger.mode
dmm.measure.analogtrigger.mode = setting
setting = dmm.measure.getattribute(function, dmm.ATTR_MEAS_ATRIG_MODE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_ATRIG_MODE, setting)
channel.setdmm("channelList", dmm.ATTR_MEAS_ATRIG_MODE, setting)
setting = channel.getdmm("channelList", dmm.ATTR_MEAS_ATRIG_MODE)

```

<i>setting</i>	The mode setting: - Edge (signal crosses one level): <code>dmm.MODE_EDGE</code> - Window (signal enters or exits a window defined by two levels): <code>dmm.MODE_WINDOW</code> - No analog triggering: <code>dmm.MODE_OFF</code>
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

<code>dmm.FUNC_DC_VOLTAGE</code>	<code>dmm.FUNC_RESISTANCE</code>	<code>dmm.FUNC_ACV_FREQUENCY</code>
<code>dmm.FUNC_AC_VOLTAGE</code>	<code>dmm.FUNC_4W_RESISTANCE</code>	<code>dmm.FUNC_ACV_PERIOD</code>
<code>dmm.FUNC_DC_CURRENT</code>	<code>dmm.FUNC_DIODE</code>	<code>dmm.FUNC_DCV_RATIO</code>
<code>dmm.FUNC_AC_CURRENT</code>	<code>dmm.FUNC_CAPACITANCE</code>	<code>dmm.FUNC_DIGITIZE_CURRENT</code>
<code>dmm.FUNC_TEMPERATURE</code>	<code>dmm.FUNC_CONTINUITY</code>	<code>dmm.FUNC_DIGITIZE_VOLTAGE</code>

Details

When edge is selected, the analog trigger occurs when the signal crosses a certain level. You also specify if the analog trigger occurs on the rising or falling edge of the signal.

When window is selected, the analog trigger occurs when the signal enters or exits the window defined by the low and high signal levels.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_CURRENT
dmm.measure.range = 3
dmm.measure.autozero.enable = dmm.OFF
dmm.measure.analogtrigger.mode = dmm.MODE_EDGE
dmm.measure.analogtrigger.edge.level = 2.5
dmm.measure.analogtrigger.edge.slope = dmm.SLOPE_FALLING
```

Set measure function to DC current.
Set range to 3 A.
Disable autozero.
Set the analog trigger mode to edge.
Set the analog trigger level to 2.5 A.
Set the level to be detected on a falling edge.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 3)
channel.setdmm("101:109", dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_MODE, dmm.MODE_EDGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE, dmm.SLOPE_FALLING)
```

For channels 1 through 9 on slot 1, set the measure function to DC voltage.
Set range to 3 V.
Disable autozero.
Set the analog trigger mode to edge.
Set the analog trigger level to 5 V.
Set the level to be detected on a falling edge.

Also see

[Analog triggering example with digitize function](#) (on page 8-22)

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.mode](#) (on page 14-140)

dmm.measure.analogtrigger.window.direction

This attribute defines if the analog trigger occurs when the signal enters or leaves the defined high and low analog signal level boundaries.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.DIRECTION_ENTER

Usage

```
value = dmm.measure.analogtrigger.window.direction
dmm.measure.analogtrigger.window.direction = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_WINDOW_DIRECTION)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_WINDOW_DIRECTION, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_ATRIG_WINDOW_DIRECTION, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_ATRIG_WINDOW_DIRECTION)
```

<i>value</i>	The direction: ■ Enter: dmm.DIRECTION_ENTER ■ Leave: dmm.DIRECTION_LEAVE
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This is only available when the analog trigger mode is set to window.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.range = 10
dmm.measure.autozero.enable = dmm.OFF
dmm.measure.analogtrigger.mode = dmm.MODE_WINDOW
dmm.measure.analogtrigger.window.levelhigh = 5
dmm.measure.analogtrigger.window.levellow = 1
dmm.measure.analogtrigger.window.direction = dmm.DIRECTION_LEAVE
```

Set measure function to DC voltage.
Set range to 10 V.
Disable autozero.
Set the analog trigger mode to window.
Set the analog trigger high level to 5 V.
Set the analog trigger low level to 1 V.
Set the trigger to occur when the signal leaves the window.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_MODE, dmm.MODE_WINDOW)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_HIGH, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_LOW, 1)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_DIRECTION,
dmm.DIRECTION_LEAVE)
```

For channels 1 through 9 on slot 1, set the measure function to DC voltage.
Set range to 10 V.
Disable autozero.
Set the analog trigger mode to window.
Set the analog trigger high level to 5 V.
Set the analog trigger low level to 1 V.
Set the trigger to occur when the signal leaves the window.

Also see

[dmm.digitize.analogtrigger.window.direction](#) (on page 14-142)
[dmm.measure.analogtrigger.mode](#) (on page 14-195)
[dmm.measure.analogtrigger.window.levelhigh](#) (on page 14-199)
[dmm.measure.analogtrigger.window.levellow](#) (on page 14-201)

dmm.measure.analogtrigger.window.levelhigh

This attribute defines the upper boundary of the analog trigger window.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	DC current: 5 μ A DC voltage: 50 mV

Usage

```
value = dmm.measure.analogtrigger.window.levelhigh
dmm.measure.analogtrigger.window.levelhigh = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_WINDOW_LEVEL_HIGH)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_WINDOW_LEVEL_HIGH, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_HIGH, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_HIGH)
```

<i>value</i>	The upper boundary of the window
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Only available when the analog trigger mode is set to window.

The high level must be greater than the low level.

To use the analog trigger with the measure functions, a range must be set (you cannot use autorange) and autozero must be disabled.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.range = 10
dmm.measure.autozero.enable = dmm.OFF
dmm.measure.analogtrigger.mode = dmm.MODE_WINDOW
dmm.measure.analogtrigger.window.levelhigh = 5
dmm.measure.analogtrigger.window.levellow = 1
dmm.measure.analogtrigger.window.direction = dmm.DIRECTION_LEAVE
```

Set measure function to DC voltage.
Set range to 10 V.
Disable autozero.
Set the analog trigger mode to window.
Set the analog trigger high level to 5 V.
Set the analog trigger low level to 1 V.
Set the trigger to occur when the signal leaves the window.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_MODE, dmm.MODE_WINDOW)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_HIGH, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_LOW, 1)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_DIRECTION,
    dmm.DIRECTION_LEAVE)
```

For channels 1 through 9 on slot 1, set the measure function to DC voltage.

Set range to 10 V.

Disable autozero.

Set the analog trigger mode to window.

Set the analog trigger high level to 5 V.

Set the analog trigger low level to 1 V.

Set the trigger to occur when the signal leaves the window.

Also see

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.window.levelhigh](#) (on page 14-144)

[dmm.measure.analogtrigger.mode](#) (on page 14-195)

[dmm.measure.analogtrigger.window.direction](#) (on page 14-197)

[dmm.measure.analogtrigger.window.levellow](#) (on page 14-201)

dmm.measure.analogtrigger.window.levellow

This attribute defines the lower boundary of the analog trigger window.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0

Usage

```
value = dmm.measure.analogtrigger.window.levellow
dmm.measure.analogtrigger.window.levellow = value
value = dmm.measure.getattribute(function, dmm.ATTR_DIGI_WINDOW_LEVEL_LOW)
dmm.measure.setattribute(function, dmm.ATTR_DIGI_WINDOW_LEVEL_LOW, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_ATTRIG_WINDOW_LEVEL_LOW, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_ATTRIG_WINDOW_LEVEL_LOW)
```

<i>value</i>	The lower boundary of the window
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Only available when the analog trigger mode is set to window.

The high level must be greater than the low level.

To use the analog trigger with the measure functions, a range must be set (you cannot use autorange) and autozero must be disabled.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.range = 10
dmm.measure.autozero.enable = dmm.OFF
dmm.measure.analogtrigger.mode = dmm.MODE_WINDOW
dmm.measure.analogtrigger.window.levelhigh = 5
dmm.measure.analogtrigger.window.levellow = 1
dmm.measure.analogtrigger.window.direction = dmm.DIRECTION_LEAVE
```

Set measure function to DC voltage.
Set range to 10 V.
Disable autozero.
Set the analog trigger mode to window.
Set the analog trigger high level to 5 V.
Set the analog trigger low level to 1 V.
Set the trigger to occur when the signal leaves the window.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_MODE, dmm.MODE_WINDOW)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_HIGH, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_LOW, 1)
channel.setdmm("101:109", dmm.ATTR_MEAS_ATRIG_WINDOW_DIRECTION,
    dmm.DIRECTION_LEAVE)
```

For channels 1 through 9 on slot 1, set the measure function to DC voltage.

Set range to 10 V.

Disable autozero.

Set the analog trigger mode to window.

Set the analog trigger high level to 5 V.

Set the analog trigger low level to 1 V.

Set the trigger to occur when the signal leaves the window.

Also see

[Analog triggering overview](#) (on page 8-20)

[dmm.digitize.analogtrigger.window.levellow](#) (on page 14-146)

[dmm.measure.analogtrigger.mode](#) (on page 14-195)

[dmm.measure.analogtrigger.window.direction](#) (on page 14-197)

[dmm.measure.analogtrigger.window.levelhigh](#) (on page 14-199)

dmm.measure.aperture

This function determines the aperture setting for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	See Details

Usage

```
value = dmm.measure.aperture
dmm.measure.aperture = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_APERTURE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_APERTURE, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_APERTURE, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_APERTURE)
```

<i>value</i>	The integration rate; see Details for ranges
<i>function</i>	The measure function; see Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Function	Default value	Range
Voltage (AC and DC)	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Current (AC and DC)	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Resistance (2-wire and 4-wire)	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Diode	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Temperature	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s
Frequency and Period	200 ms	2 ms to 273 ms
Voltage ratio	60 Hz: 16.67 ms 50 Hz: 20 ms	8.333 μ s to 0.25 s 10 μ s to 0.24 s

The aperture sets the amount of time the ADC takes when making a measurement, which is the integration period for the selected measurement function. The integration period is specified in seconds. In general, a short integration period provides a fast reading rate, while a long integration period provides better accuracy. The selected integration period is a compromise between speed and accuracy.

During the integration period, if an external trigger with a count of 1 is sent, the trigger is ignored. If the count is set to more than 1, the first reading is initialized by this trigger. Subsequent readings occur as rapidly as the instrument can make them. If a trigger occurs during the group measurement, the trigger is latched and another group of measurements with the same count will be triggered after the current group completes.

You can also set the integration rate by setting the number of power-line cycles (NPLCs). Changing the NPLC value changes the aperture time and changing the aperture time changes the NPLC value.

To calculate the aperture based on the NPLC value, use the following formula.

$$\text{Aperture} = \frac{\text{NPLC}}{f}$$

where:

- Aperture is the integration rate in seconds for each integration
- NPLC is the number of power-line cycles for each integration
- f is the power-line frequency

If you set the NPLCs, the aperture setting changes to reflect that value. If you set the aperture, the NPLC setting is changed.

For the AC voltage and AC current functions, the aperture value is fixed and cannot be changed.

If line synchronization is enabled, the integration period does not start until the beginning of the next power-line cycle. For example, if a reading is triggered at the positive peak of a power-line cycle, the integration period does not start until that power-line cycle is completed. The integration period starts when the positive-going sine wave crosses 0 volts.

To see the line frequency that is auto-detected by the instrument, use the `localnode.linefreq` command.

Example 1

```
dmm.measure.aperture = 0.0035
```

Set the aperture to 3.5 ms.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_APERTURE, 0.0035)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.

Set the aperture to 3.5 ms.

Also see

- [dmm.digitize.aperture](#) (on page 14-148)
- [dmm.measure.linesync](#) (on page 14-257)
- [dmm.measure.nplc](#) (on page 14-269)
- [localnode.linefreq](#) (on page 14-357)

dmm.measure.autodelay

This attribute enables or disables the automatic delay that occurs before each measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.DELAY_ON

Usage

```
value = dmm.measure.autodelay
dmm.measure.autodelay = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_AUTO_DELAY)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_AUTO_DELAY, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_AUTO_DELAY, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_AUTO_DELAY)
```

value	Enable the delay: dmm.DELAY_ON Disable the delay: dmm.DELAY_OFF
channelList	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

When this is enabled, a delay is added after a range or function change to allow the instrument to settle.

Example 1

```
dmm.measure.func = dmm.FUNC_RESISTANCE
dmm.measure.autodelay = dmm.DELAY_ON
dmm.measure.count = 10
ReadingBufferOne = buffer.make(1000)
dmm.measure.read(ReadingBufferOne)
```

Set the instrument to measure 2-wire ohms.
Turn automatic delay on.
Create a buffer named ReadingBufferOne.
Set the number of measurements to 10.
Make 10 measurements and store them in the reading buffer.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_AUTO_DELAY, dmm.DELAY_OFF)
channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_5_5)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.
Set the range to 5 V, which selects the 10 V range.
Set auto delay off.

Also see

[channel.getdmm](#) (on page 14-75)

[channel.setdmm](#) (on page 14-88)

[delay\(\)](#) (on page 14-112)

dmm.measure.autozero.enable

This attribute enables or disables automatic updates to the internal reference measurements (autozero) of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.ON

Usage

```
state = dmm.measure.autozero.enable
dmm.measure.autozero.enable = state
state = dmm.measure.getattribute(function, dmm.ATTR_MEAS_AUTO_ZERO)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_AUTO_ZERO, state)
channel.setdmm("channelList", dmm.ATTR_MEAS_AUTO_ZERO, state)
state = channel.getdmm("channelList", dmm.ATTR_MEAS_AUTO_ZERO)
```

<i>state</i>	Disable autozero: dmm.OFF Enable autozero: dmm.ON
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

To ensure the accuracy of readings, the instrument must periodically get new measurements of its internal ground and voltage reference. The time interval between updates to these reference measurements is determined by the integration aperture that is being used for measurements. The DAQ6510 uses separate reference and zero measurements for each aperture.

By default, the instrument automatically checks these reference measurements whenever a signal measurement is made.

The time to make the reference measurements is in addition to the normal measurement time. If timing is critical, you can disable autozero to avoid this time penalty.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.autozero.enable = dmm.OFF
```

Set autozero off for voltage measurements.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_AUTO_ZERO, dmm.OFF)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 0.5)
channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_5_5)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.

Set autozero off.

Set NPLC to 0.5.

Set the number of display digits to 5½.

Also see

[Automatic reference measurements](#) (on page 4-53)

[channel.getdmm](#) (on page 14-75)

[channel.setdmm](#) (on page 14-88)

[dmm.measure.autozero.once\(\)](#) (on page 14-211)

[dmm.measure.nplc](#) (on page 14-269)

dmm.measure.autorange

This attribute determines if the measurement range is set manually or automatically for the selected measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.ON

Usage

```
state = dmm.measure.autorange
dmm.measure.autorange = state
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_RANGE_AUTO)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_RANGE_AUTO, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_RANGE_AUTO, state)
state = channel.getdmm("channelList", dmm.ATTR_MEAS_RANGE_AUTO)
```

<i>state</i>	Set the measurement range manually: <code>dmm.OFF</code> Set the measurement range automatically: <code>dmm.ON</code>
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

<code>dmm.FUNC_DC_VOLTAGE</code>	<code>dmm.FUNC_RESISTANCE</code>	<code>dmm.FUNC_ACV_FREQUENCY</code>
<code>dmm.FUNC_AC_VOLTAGE</code>	<code>dmm.FUNC_4W_RESISTANCE</code>	<code>dmm.FUNC_ACV_PERIOD</code>
<code>dmm.FUNC_DC_CURRENT</code>	<code>dmm.FUNC_DIODE</code>	<code>dmm.FUNC_DCV_RATIO</code>
<code>dmm.FUNC_AC_CURRENT</code>	<code>dmm.FUNC_CAPACITANCE</code>	<code>dmm.FUNC_DIGITIZE_CURRENT</code>
<code>dmm.FUNC_TEMPERATURE</code>	<code>dmm.FUNC_CONTINUITY</code>	<code>dmm.FUNC_DIGITIZE_VOLTAGE</code>

Details

This command determines how the range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was last selected by autorange.

When this command is set to on, the instrument automatically goes to the most sensitive range to perform the measurement.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Autorange selects the best range in which to measure the signal that is applied to the input terminals of the instrument. When autorange is enabled, the range increases at 120 percent of range. The range decreases occur when the reading is <10 percent of nominal range. For example, if you are on the 1 V range and autorange is enabled, the instrument autoranges up to the 10 V range when the measurement exceeds 1.2 V. It autoranges down to the 100 mV range when the measurement falls below 1 V.

Example 1

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

Select the measurement function to be DC voltage. Set autorange on.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)  
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE_AUTO, dmm.ON)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.
Set the channels to autorange.

Also see

[channel.getdmm](#) (on page 14-75)
[channel.setdmm](#) (on page 14-88)
[dmm.measure.range](#) (on page 14-275)
[Ranges](#) (on page 4-54)

dmm.measure.autozero.once()

This function causes the instrument to refresh the reference and zero measurements once.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

dmm.measure.autozero.once()

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command forces a refresh of the reference and zero measurements that are used for the present aperture setting for the selected function.

When autozero is set to off, the instrument may gradually drift out of specification. To minimize the drift, you can send the once command to make a reference and zero measurement immediately before a test sequence.

If the NPLC setting is less than 0.2 PLC, sending autozero once can result in delay of more than a second.

Example

dmm.measure.autozero.once()	Do a one-time refresh of the reference and zero measurements.

Also see

- [Automatic reference measurements](#) (on page 4-53)
- [dmm.measure.autozero.enable](#) (on page 14-207)

dmm.measure.bias.level

This attribute selects the amount of current the instrument sources when it makes measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0.001 (1 mA)

Usage

```
value = dmm.measure.bias.level
dmm.measure.bias.level = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_BIAS_LEVEL)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_BIAS_LEVEL, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_BIAS_LEVEL, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_BIAS_LEVEL)
```

<i>value</i>	Enter the value: 10 μA: 1e-5 100 μA: 0.0001 1 mA: 0.001 10 mA: 0.01
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Selects the amount of current that is sourced by the instrument to make measurements.

Example 1

```
dmm.measure.bias.level = 0.0001
```

Set a bias level of 100 μ A.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DIODE)
channel.setdmm("101:109", dmm.ATTR_MEAS_BIAS_LEVEL, 0.0001)
```

For channels 1 through 9 on slot 1, set the DMM function to diode.
Set the bias level to 100 μ A.

Also see

None

dmm.measure.configlist.catalog()

This function returns the name of one measure configuration list that is stored on the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.measure.configlist.catalog()
```

Details

You can use this command to retrieve the names of measure configuration lists that are stored in the instrument.

This command returns one name each time you send it. This command returns `nil` to indicate that there are no more names to return. If the command returns `nil` the first time you send it, no measure configuration lists have been created for the instrument.

Example

<pre>print(dmm.measure.configlist.catalog())</pre>	Request the name of one measure configuration list that is stored in the instrument. Send the command again until it returns <code>nil</code> to get all stored lists.
<pre>print(dmm.measure.configlist.catalog())</pre>	If there are two configuration lists on the instrument. Example output: <code>testMeasList</code>
<pre>print(dmm.measure.configlist.catalog())</pre>	<code>myMeasList</code>
<pre>print(dmm.measure.configlist.catalog())</pre>	<code>nil</code>

Also see

- [Configuration lists](#) (on page 4-99)
- [createconfigscript\(\)](#) (on page 14-105)
- [dmm.measure.configlist.create\(\)](#) (on page 14-214)

dmm.measure.configlist.create()

This function creates an empty measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.measure.configlist.create("listName")
```

<i>listName</i>	A string that represents the name of a measure configuration list

Details

This command creates an empty configuration list. To add configuration indexes to this list, you need to use the store command.

Configuration lists are not saved when the instrument is turned off. To save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Example

<pre>dmm.measure.configlist.create("MyMeasList")</pre>
Create a measure configuration list named MyMeasList.

Also see

- [Configuration lists](#) (on page 4-99)
- [dmm.measure.configlist.catalog\(\)](#) (on page 14-213)
- [dmm.measure.configlist.delete\(\)](#) (on page 14-215)
- [dmm.measure.configlist.query\(\)](#) (on page 14-216)
- [dmm.measure.configlist.recall\(\)](#) (on page 14-217)
- [dmm.measure.configlist.size\(\)](#) (on page 14-218)
- [dmm.measure.configlist.store\(\)](#) (on page 14-218)

dmm.measure.configlist.delete()

This function deletes a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.measure.configlist.delete("listName")
dmm.measure.configlist.delete("listName", index)
```

<i>listName</i>	A string that represents the name of a measure configuration list
<i>index</i>	A number that defines a specific configuration index in the configuration list

Details

Deletes a configuration list. If the index is not specified, the entire configuration list is deleted. If the index is specified, only the specified configuration index in the list is deleted.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and the all the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the higher numbered index first, then the next lower index, and so on. This also means that if you want to delete all the indexes in a configuration list, you must delete index 1 repeatedly until all indexes have been removed.

Example

dmm.measure.configlist.delete("myMeasList")	Delete a measure configuration list named myMeasList.
dmm.measure.configlist.delete("myMeasList", 2)	Delete configuration index 2 from the measure configuration list named myMeasList.

Also see

- [Configuration lists](#) (on page 4-99)
- [createconfigscript\(\)](#) (on page 14-105)
- [dmm.measure.configlist.create\(\)](#) (on page 14-214)

dmm.measure.configlist.query()

This function returns a list of TSP commands and parameter settings that are stored in the specified configuration index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.measure.configlist.query("listName", index)
dmm.measure.configlist.query("listName", index, "fieldSeparator")
```

listName	A string that represents the name of a measure configuration list
index	A number that defines a specific configuration index in the configuration list
fieldSeparator	String that represents the separator for the data; use one of the following: - Comma (default): , - Semicolon: ; - New line: \n

Details

This command recalls data for one configuration index.

Example

```
print(dmm.measure.configlist.query("testMeasList", 2, "\n"))
```

Returns the TSP commands and parameter settings that represent the settings in configuration index 2.

Partial example output:

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.unit = dmm.UNIT_VOLT
dmm.measure.range = 0.1
dmm.measure.autorange = dmm.ON
dmm.measure.transducer is not used
dmm.measure.detectorbandwidth is not used
dmm.measure.autozero.enable = dmm.ON
dmm.measure.autodelay = dmm.DELAY_ON
dmm.measure.displaydigits = dmm.DIGITS_6_5
dmm.measure.dbreference = 1
dmm.measure.filter.enable = dmm.OFF
dmm.measure.filter.count = 10
dmm.measure.filter.type = dmm.FILTER_REPEAT_AVG
dmm.measure.filter.window = 0
```

Also see

- [Configuration lists](#) (on page 4-99)
- [createconfigscript\(\)](#) (on page 14-105)
- [dmm.measure.configlist.create\(\)](#) (on page 14-214)

dmm.measure.configlist.recall()

This function recalls a configuration index in a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.measure.configlist.recall("listName")
dmm.measure.configlist.recall("listName", index)
```

<i>listName</i>	A string that represents the name of a measure configuration list
<i>index</i>	A number that defines a specific configuration index in the measure configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a measure configuration list. If you do not specify an index when you send the command, it recalls the settings stored in the first configuration index in the specified measure configuration list.

If you recall an invalid index (for example, calling index 3 when there are only two indexes in the configuration list) or try to recall an index from an empty configuration list, an error message is displayed.

Each index contains the settings for the selected function of that index. Settings for other functions are not affected when the configuration list index is recalled. A single index stores the settings associated with a single measure or digitize function.

This command recalls data for one configuration index.

Example

<code>dmm.measure.configlist.recall("MyMeasList")</code>	Because an index was not specified, this command recalls configuration index 1 from a configuration list named <code>MyMeasList</code> .
<code>dmm.measure.configlist.recall("MyMeasList", 5)</code>	Recalls configuration index 5 in a configuration list named <code>MyMeasList</code> .

Also see

- [Configuration lists](#) (on page 4-99)
- [createconfigscript\(\)](#) (on page 14-105)
- [dmm.measure.configlist.create\(\)](#) (on page 14-214)
- [dmm.measure.configlist.store\(\)](#) (on page 14-218)

dmm.measure.configlist.size()

This function returns the size (number of configuration indexes) of a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
indexCount = dmm.measure.configlist.size("listName")
```

<i>indexCount</i>	A number that represents the total count of indexes stored in the specified measure configuration list
<i>listName</i>	A string that represents the name of a measure configuration list

Details

This command returns the size (number of configuration indexes) of a measure configuration list.

The size of the list is equal to the number of configuration indexes in a configuration list.

Example

```
print(dmm.measure.configlist.size("testMeasList"))
```

Returns the number of configuration indexes in a measure configuration list named `testMeasList`.

Example output:

```
1
```

Also see

[Configuration lists](#) (on page 4-99)

[createconfigscript\(\)](#) (on page 14-105)

[dmm.measure.configlist.create\(\)](#) (on page 14-214)

dmm.measure.configlist.store()

This function stores the active measure or digitize settings into the named configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.measure.configlist.store("listName")  
dmm.measure.configlist.store("listName", index)
```

<i>listName</i>	A string that represents the name of a measure configuration list
<i>index</i>	A number that defines a specific configuration index in the configuration list

Details

Use this command to store the active measure or digitize settings to a configuration index in a configuration list. If the index parameter is not provided, the new settings are appended to the end of the list. The index only stores the active settings for a single active measure or digitize function.

A measure configuration list can store measure or digitize settings, but not at the same time. If the active function is a digitize function, digitize settings are saved. When the index is queried, digitize settings and their values are listed, but measure settings are listed as not being used. Similarly, if the active function is a measure function, measure settings are saved. When the index is queried, the measure settings and their values are listed, but the digitize settings are listed as not used.

Configuration lists are not saved when the instrument is turned off or reset. To save a configuration list, create a configuration script to save instrument settings, including any defined configuration lists.

Example

<code>dmm.measure.configlist.store("MyConfigList")</code>	Stores the active settings of the instrument to the end of the configuration list <code>MyConfigList</code> .
<code>dmm.measure.configlist.store("MyConfigList", 5)</code>	Stores the active settings of the instrument to configuration index 5 in the measure configuration list <code>MyConfigList</code> .

Also see

- [Configuration lists](#) (on page 4-99)
- [createconfigscript\(\)](#) (on page 14-105)
- [dmm.measure.configlist.create\(\)](#) (on page 14-214)

dmm.measure.configlist.storefunc()

This function allows you to store the settings for a measure function into a measure configuration list whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
dmm.measure.configlist.storefunc("listName", function)
dmm.measure.configlist.storefunc("listName", function, index)
```

<i>listName</i>	Name of the configuration list in which to store the function settings
<i>function</i>	The measure function settings to save into the configuration list; see Details
<i>index</i>	The number of the configuration list index in which to store the settings

Details

You must create the configuration list before using this command.

If *index* is not specified, the settings are stored to the next available index in the configuration list.

You can set *function* to the values in the following table.

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Example

```
dmm.measure.configlist.create("MyMeasList")
dmm.measure.configlist.storefunc("MyMeasList", dmm.FUNC_DC_VOLTAGE)
dmm.measure.configlist.storefunc("MyMeasList", dmm.FUNC_DC_TEMPERATURE, 2)
```

Create a measure configuration list named MyMeasList.
Store the attributes for the DC Voltage settings in index 1.
Stores attributes for the Temperature function in index 2.

Also see

[dmm.measure.configlist.create\(\)](#) (on page 14-214)
[dmm.measure.setattribute\(\)](#) (on page 14-299)

dmm.measure.count

This attribute sets the number of measurements to make when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1

Usage

```
count = dmm.measure.count
dmm.measure.count = count
count = dmm.measure.getattribute(function, dmm.ATTR_MEAS_COUNT)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_COUNT, count)
channel.setdmm("channelList", dmm.ATTR_MEAS_COUNT, count)
count = channel.getdmm("channelList", dmm.ATTR_MEAS_COUNT)
```

<i>count</i>	The number of measurements to make when a measurement is requested (maximum 1,000,000 or buffer capacity)
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command sets the number of measurements that are made when a measurement is requested. This command does not affect the trigger model.

When `dmm.measure.count` or if the function for `dmm.measure.setattribute` is the active function, this command sets the count for all measure functions. When you send `dmm.measure.setattribute` for a function that is not active, only the count for the specified function is changed.

If you set the count to a value that is larger than the capacity of the reading buffer and the buffer fill mode is set to continuous, the buffer wraps until the number of readings specified have occurred. The earliest readings in the count are overwritten. If the buffer is set to fill once, readings stop when the buffer is filled, even if the count is not complete.

NOTE

To get better performance from the instrument, use the SimpleLoop trigger-model template instead of using the count command.

Example 1

```
dmm.measure.count = 10  
dmm.measure.read()
```

Set the instrument to make 10 measurements.
Request 10 measurements.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)  
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 5)  
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 0.5)  
channel.setdmm("101:109", dmm.ATTR_MEAS_COUNT, 5)
```

For channels 1 through 10, set the DMM function to DC voltage.
Set the range to 5 V, which selects the 10 V range.
Set NPLC to 0.5.
Set the number of measurements to 5.

Also see

[dmm.digitize.count](#) (on page 14-150)
[dmm.measure.read\(\)](#) (on page 14-278)
[dmm.measure.readwithtime\(\)](#) (on page 14-279)
[trigger.model.load\(\) — SimpleLoop](#) (on page 14-487)

dmm.measure.dbreference

This attribute defines the decibel (dB) reference setting for the DMM in volts.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1 (1 V)

Usage

```
value = dmm.measure.dbreference
dmm.measure.dbreference = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_DB_REFERENCE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_DB_REFERENCE, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_DB_REFERENCE, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_DB_REFERENCE)
```

<i>value</i>	The decibel reference range: - DC voltage: 1e-7 V to 1000 V - AC voltage: 1e-7 V to 750 V
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This value only applies when the unit setting for the function is set to decibels.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.unit = dmm.UNIT_DB
dmm.measure.dbreference = 5
```

Sets the units to decibel and sets the dB reference to 5 for DC volts.

Example 2

```
channel.setdmm("201:210", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("201:210", dmm.ATTR_MEAS_UNIT, dmm.UNIT_DB)
channel.setdmm("201:210", dmm.ATTR_MEAS_DB_REFERENCE, 5)
```

For channels 1 through 10 on slot 2, set the DMM function to DC voltage.
Set the units to decibels.
Set the dB reference to 5 V.

Also see

[channel.getdmm](#) (on page 14-75)

[channel.setdmm](#) (on page 14-88)

[dmm.digitize.dbreference](#) (on page 14-152)

[dmm.measure.dbmreference](#) (on page 14-225)

[dmm.measure.unit](#) (on page 14-324)

dmm.measure.dbmreference

This attribute defines the decibel-milliwatts (dBm) reference.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	75 (75 Ω)

Usage

```
value = dmm.measure.dbmreference
dmm.measure.dbmreference = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_DBM_REFERENCE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_DBM_REFERENCE, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_DBM_REFERENCE, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_DBM_REFERENCE)
```

<i>value</i>	The dBm impedance value (1 Ω to 9999 Ω)
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This value only applied when the unit setting for the function is set to dBm.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.unit = dmm.UNIT_DBM
dmm.measure.dbmreference = 85
```

Sets the units to dBm and sets the dBm reference to 85 Ω .

Example 2

```
channel.setdmm("201:210", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("201:210", dmm.ATTR_MEAS_UNIT, dmm.UNIT_DBM)
channel.setdmm("201:210", dmm.ATTR_MEAS_DBM_REFERENCE, 85)
```

For channels 1 through 10 on slot 2, set the DMM function to DC voltage.
Set the units to decibel-milliwatts.
Set the dBm reference to 85 Ω .

Also see

[channel.getdmm](#) (on page 14-75)
[channel.setdmm](#) (on page 14-88)

[dmm.digitize.dbmreference](#) (on page 14-153)

[dmm.measure.dbreference](#) (on page 14-223)

[dmm.measure.unit](#) (on page 14-324)

dmm.measure.detectorbandwidth

This attribute selects the detector bandwidth for AC current and AC voltage measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.DETECTBW_3HZ

Usage

```
value = dmm.measure.detectorbandwidth
dmm.measure.detectorbandwidth = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_DETECTBW)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_DETECTBW, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_DETECTBW, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_DETECTBW)
```

<i>value</i>	The bandwidth that is closest to the line frequency: 3 Hz: dmm.DETECTBW_3HZ 30 Hz: dmm.DETECTBW_30HZ 300 Hz: dmm.DETECTBW_300HZ
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

You can set the detector bandwidth to improve measurement accuracy. Select the bandwidth that contains the lowest frequency component of the input signal. For example, if the lowest frequency component of your input signal is 40 Hz, use a bandwidth setting of 30 Hz.

Example 1

```
dmm.measure.func = dmm.FUNC_AC_CURRENT
dmm.measure.detectorbandwidth = dmm.DETECTBW_3HZ
```

Set the measure function to AC current.
Set the bandwidth to 3 Hz.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_AC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_DETECTBW, dmm.DETECTBW_3HZ)
```

For channels 1 through 9 on slot 1, set the DMM function to AC voltage. Set the measure function to AC current.

Also see

[dmm.measure.autozero.enable](#) (on page 14-207)

dmm.measure.displaydigits

This attribute determines the number of digits that are displayed for measurements on the front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	See Functions and defaults

Usage

```
value = dmm.measure.displaydigits
dmm.measure.displaydigits = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_DIGITS)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_DIGITS, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_DIGITS, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_DIGITS)
```

<i>value</i>	3.5 digit resolution: dmm.DIGITS_3_5 4.5 digit resolution: dmm.DIGITS_4_5 5.5 digit resolution: dmm.DIGITS_5_5 6.5 digit resolution: dmm.DIGITS_6_5
<i>function</i>	See Functions and defaults
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions and defaults

Function	Def	Function	Def	Function	Def
dmm.FUNC_DC_VOLTAGE	6	dmm.FUNC_RESISTANCE	6	dmm.FUNC_ACV_FREQUENCY	6
dmm.FUNC_AC_VOLTAGE	6	dmm.FUNC_4W_RESISTANCE	6	dmm.FUNC_ACV_PERIOD	6
dmm.FUNC_DC_CURRENT	6	dmm.FUNC_DIODE	6	dmm.FUNC_DCV_RATIO	6
dmm.FUNC_AC_CURRENT	6	dmm.FUNC_CAPACITANCE	4	dmm.FUNC_DIGITIZE_CURRENT	4
dmm.FUNC_TEMPERATURE	3	dmm.FUNC_CONTINUITY	4	dmm.FUNC_DIGITIZE_VOLTAGE	4

Details

This command affects how the reading for a measurement is displayed on the front panel of the instrument. It does not affect the number of digits returned in a remote command reading. It also does not affect the accuracy or speed of measurements.

The display digits setting is saved with the function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

The change in digits occurs the next time a measurement is made.

To change the number of digits returned in a remote command reading, use `format.asciiprecision`.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.displaydigits = dmm.DIGITS_5_5
```

Set the measurement function to voltage with a front-panel display resolution of 5½.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 0.5)
channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_5_5)
```

For channels 1 through 9, set the DMM function to DC voltage.
Set the range to 5 V, which selects the 10 V range.
Set NPLC to 0.5.
Set the number of display digits to 5½.

Also see

[channel.getdmm](#) (on page 14-75)
[channel.setdmm](#) (on page 14-88)
[dmm.digitize.displaydigits](#) (on page 14-154)
[format.asciiprecision](#) (on page 14-341)

dmm.measure.filter.count

This attribute sets the number of measurements that are averaged when filtering is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	10

Usage

```
filterCount = dmm.measure.filter.count
dmm.measure.filter.count = filterCount
filterCount = dmm.measure.getattribute(function, dmm.ATTR_MEAS_FILTER_COUNT)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_FILTER_COUNT, filterCount)
channel.setdmm("channelList", dmm.ATTR_MEAS_FILTER_COUNT, filterCount)
filterCount = channel.getdmm("channelList", dmm.ATTR_MEAS_FILTER_COUNT)
```

<i>filterCount</i>	The number of readings required for each filtered measurement (1 to 100)
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The filter count is the number of readings that are acquired and stored in the filter stack for the averaging calculation. When the filter count is larger, more filtering is done, and the data is less noisy.

Example 1

<pre>dmm.measure.func = dmm.FUNC_DC_CURRENT dmm.measure.filter.count = 10 dmm.measure.filter.type = dmm.FILTER_MOVING_AVG dmm.measure.filter.enable = dmm.ON</pre>	Set the measurement function to current. Set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_COUNT, 10)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_TYPE, dmm.FILTER_REPEAT_AVG)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_WINDOW, 0.25)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_ENABLE, dmm.ON)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.
Set the averaging filter type to repeating with a filter count of 10.
Set the filter window to 0.25 and enable the averaging filter.

Also see

[Filtering measurement data](#) (on page 4-65)

[dmm.measure.filter.enable](#) (on page 14-232)

[dmm.measure.filter.type](#) (on page 14-234)

dmm.measure.filter.enable

This attribute enables or disables the averaging filter for measurements of the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OFF

Usage

```
filterState = dmm.measure.filter.enable
dmm.measure.filter.enable = filterState
filterState = dmm.measure.getattribute(function, dmm.ATTR_MEAS_FILTER_ENABLE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_FILTER_ENABLE, filterState)
channel.setdmm("channelList", dmm.ATTR_MEAS_FILTER_ENABLE, filterState)
filterState = channel.getdmm("channelList", dmm.ATTR_MEAS_FILTER_ENABLE)
```

<i>filterState</i>	The filter status: - Disable the filter: dmm.OFF - Enable the filter: dmm.ON
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command enables or disables the averaging filter. When this is enabled, the reading returned by the instrument is an averaged value, taken from multiple measurements. The settings of the filter count and filter type for the selected measure function determines how the reading is averaged.

Example 1

<pre>dmm.measure.func = dmm.FUNC_DC_CURRENT dmm.measure.filter.count = 10 dmm.measure.filter.type = dmm.FILTER_MOVING_AVG dmm.measure.filter.enable = dmm.ON</pre>	Set the measurement function to current. Set the averaging filter type to moving average, with a filter count of 10. Enable the averaging filter.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_COUNT, 10)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_TYPE, dmm.FILTER_REPEAT_AVG)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_WINDOW, 0.25)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_ENABLE, dmm.ON)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.
Set the averaging filter type to repeating with a filter count of 10.
Set the filter window to 0.25 and enable the averaging filter.

Also see

[Filtering measurement data](#) (on page 4-65)

[dmm.measure.filter.count](#) (on page 14-230)

[dmm.measure.filter.type](#) (on page 14-234)

dmm.measure.filter.type

This attribute defines the type of averaging filter that is used for the selected measure function when the measurement filter is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.FILTER_REPEAT_AVG

Usage

```
type = dmm.measure.filter.type
dmm.measure.filter.type = type
type = dmm.measure.getattribute(function, dmm.ATTR_MEAS_FILTER_TYPE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_FILTER_TYPE, type)
channel.setdmm("channelList", dmm.ATTR_MEAS_FILTER_TYPE, type)
type = channel.getdmm("channelList", dmm.ATTR_MEAS_FILTER_TYPE)
```

<i>type</i>	The filter type setting: - Repeating filter: dmm.FILTER_REPEAT_AVG - Moving filter: dmm.FILTER_MOVING_AVG (not available for channels) - Hybrid filter: dmm.FILTER_HYBRID_AVG
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

When the repeating average filter is selected, a set of measurements are made. These measurements are stored in a measurement stack and averaged together to produce the averaged sample. Once the averaged sample is produced, the stack is flushed, and the next set of data is used to produce the next averaged sample. This type of filter is the slowest, since the stack must be completely filled before an averaged sample can be produced, but it provides more stable results.

When the moving average filter is selected, the measurements are added to the stack continuously on a first-in, first-out basis. As each measurement is made, the oldest measurement is removed from the stack. A new averaged sample is produced using the new measurement and the data that is now in the stack.

NOTE

When the moving average filter is first selected, the stack is empty. When the first measurement is made, it is copied into all the stack locations to fill the stack. A true average is not produced until the stack is filled with new measurements. The size of the stack is determined by the filter count setting.

The repeating average filter produces slower results but produces more stable results than the moving average filter. For either method, the greater the number of measurements that are averaged, the slower the averaged sample rate, but the lower the noise error. Trade-offs between speed and noise are normally required to tailor the instrumentation to your measurement application.

The hybrid average filter is only available when the buffer style is set to Full. It is similar to the moving average filter, except that it adds the number of measurements defined by the count to the stack before making the first averaged measurement. This ensures that the filter buffer is filled before returning the first measurement.

The repeating average filter is the only filter option available for use with channels.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_CURRENT
dmm.measure.filter.type = dmm.FILTER_MOVING_AVG
dmm.measure.filter.enable = dmm.ON
```

Set the measurement function to DC current.
Set the filter type to moving average and enable filtered measurements.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_COUNT, 10)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_TYPE, dmm.FILTER_REPEAT_AVG)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_WINDOW, 0.25)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_ENABLE, dmm.ON)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.
Set the averaging filter type to repeating with a filter count of 10.
Set the filter window to 0.25 and enable the averaging filter.

Also see

[dmm.measure.filter.enable](#) (on page 14-232)

dmm.measure.filter.window

This attribute sets the window for the averaging filter that is used for measurements for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0 (no filter)

Usage

```
value = dmm.measure.filter.window
dmm.measure.filter.window = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_FILTER_WINDOW)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_FILTER_WINDOW, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_FILTER_WINDOW, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_FILTER_WINDOW)
```

<i>value</i>	The filter window setting; the range is between 0 and 10 to indicate percent of range
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command selects the window size for the averaging filter.

The noise window allows a faster response time to large signal step changes. A reading that falls outside the plus or minus noise window fills the filter stack immediately.

If the noise does not exceed the selected percentage of range, the reading is based on an average of reading conversions, which is the normal averaging filter. If the noise does exceed the selected percentage, the reading is a single reading conversion, and new averaging starts from this point.

Example 1

<pre>dmm.measure.func = dmm.FUNC_RESISTANCE dmm.measure.filter.type = dmm.FILTER_MOVING_AVG dmm.measure.filter.window = 0.25 dmm.measure.filter.enable = dmm.ON</pre>	Set the measure function to 2-wire ohms. Set the filter type to moving average. Set the filter window to 0.25 and enable filtered measurements.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_COUNT, 10)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_TYPE, dmm.FILTER_REPEAT_AVG)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_WINDOW, 0.25)
channel.setdmm("101:109", dmm.ATTR_MEAS_FILTER_ENABLE, dmm.ON)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.
Set the averaging filter type to repeating with a filter count of 10.
Set the filter window to 0.25 and enable the averaging filter.

Also see

[dmm.measure.filter.enable](#) (on page 14-232)

[dmm.measure.filter.type](#) (on page 14-234)

dmm.measure.fourrtd

This attribute defines the type of 4-wire RTD that is being used

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.RTD_PT100

Usage

```
RTDType = dmm.measure.fourrtd
dmm.measure.fourrtd = RTDType
RTDType= dmm.measure.getattribute(function, dmm.ATTR_MEAS_FOUR_RTD)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_FOUR_RTD, RTDType)
channel.setdmm("channellist", dmm.ATTR_MEAS_FOUR_RTD, RTDType)
RTDType = channel.getdmm("channellist", dmm.ATTR_MEAS_FOUR_RTD)
```

<i>RTDType</i>	The type of 4-wire RTD: ■ PT100: dmm.RTD_PT100 ■ PT385: dmm.RTD_PT385 ■ PT3916: dmm.RTD_PT3916 ■ D100: dmm.RTD_D100 ■ F100: dmm.RTD_F100 ■ User-specified type: dmm.RTD_USER
<i>function</i>	See Functions
<i>channellist</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The transducer type must be set to temperature and the transducer must be set to 4-wire RTD before you can set the RTD type.

Example 1

<pre>dmm.measure.func = dmm.FUNC_TEMPERATURE dmm.measure.transducer = dmm.TRANS_FOURRTD dmm.measure.fourrtd = dmm.RTD_PT3916</pre>	Set the measure function to temperature. Set the transducer type to 4-wire RTD. Set the RTD type to PT3916.

Example 2

```
channel.setdmm("101:104", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:104", dmm.ATTR_MEAS_TRANSducer, dmm.TRANS_FOURRTD)
channel.setdmm("101:104", dmm.ATTR_MEAS_FOUR_RTD, dmm.RTD_PT3916)
```

For channels 1 through 4, set the DMM function to temperature.
Set the transducer type to 3-wire RTD. Set the RTD type to PT3916.

Also see

[dmm.measure.rtdalpha](#) (on page 14-289)

[dmm.measure.rtdbeta](#) (on page 14-291)

[dmm.measure.rtddelta](#) (on page 14-293)

[dmm.measure.rtdzero](#) (on page 14-295)

[dmm.measure.transducer](#) (on page 14-320)

[Temperature measurements](#) (on page 4-26)

dmm.measure.func

This attribute selects the active measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.FUNC_DC_VOLTAGE

Usage

```
mFunction = dmm.measure.func
dmm.measure.func = mFunction
channel.setdmm("channellist", dmm.ATTR_MEAS_FUNCTION, mFunction)
mFunction = channel.getdmm("channellist", dmm.ATTR_MEAS_FUNCTION)
```

<i>mFunction</i>	The type of measurement; see Functions for options
<i>channellist</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Set this command to the type of measurement you want to make.

Reading this command returns the measure function that is presently active.

When you select a function, settings for other commands that are related to the function become active. For example, assume that:

- You selected the current function and set the math function to reciprocal.
- You changed to the voltage function and set the math function to percent.

If you return to the current function, the math function returns to reciprocal. If you then switch from the current function to the voltage function, the math function returns to percent. All attributes that begin with `dmm.measure.` are saved with the active measure function unless otherwise indicated in the command description.

If a digitize measurement function is active, calling this command returns `dmm.FUNC_NONE`. The no function setting is automatically made if you select a function with `dmm.digitize.func` or through the options from the front-panel Digitize Functions tab.

If a channel is closed when you assign a function to the channel, all other channels are opened.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.math.format = dmm.MATH_PERCENT
dmm.measure.math.enable = dmm.ON
dmm.measure.func = dmm.FUNC_RESISTANCE
dmm.measure.math.format = dmm.MATH_RECIPROCAL
dmm.measure.math.enable = dmm.ON
print(dmm.measure.math.format)
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
print(dmm.measure.math.format)
```

Sets the instrument to measure voltage and set the math format to percent and enable the math functions.

Set the instrument to measure resistance and set the math format to reciprocal and enable the math functions.

Print the math format while the resistance measurement function is selected. The output is:

```
dmm.MATH_RECIPROCAL
```

Change the function to voltage. Print the math format. The output is:

```
dmm.MATH_PERCENT
```

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 0.5)
channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_5_5)
```

For channels 1 through 9, set the DMM function to DC voltage.

Set the range to 5 V, which selects the 10 V range.

Set NPLC to 0.5.

Set the number of display digits to 5½.

Also see

[channel.getdmm](#) (on page 14-75)

[channel.setdmm](#) (on page 14-88)

[dmm.digitize.func](#) (on page 14-155)

dmm.measure.getattribute()

This function returns the setting for a function attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = dmm.measure.getattribute(function, setting)
```

<i>value</i>	The attribute value
<i>function</i>	The measurement function; see Details
<i>setting</i>	The attribute for the function; refer to dmm.measure.setattribute() (on page 14-299) for available settings

Details

The options for *function* are shown in the following table.

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

You can retrieve one attribute at a time.

Example

```
print(dmm.measure.getattribute(dmm.FUNC_DC_VOLTAGE, dmm.ATTR_MEAS_RANGE))
print(dmm.measure.getattribute(dmm.FUNC_DC_VOLTAGE, dmm.ATTR_MEAS_NPLC))
print(dmm.measure.getattribute(dmm.FUNC_DC_VOLTAGE, dmm.ATTR_MEAS_DIGITS))
```

Retrieve the range, NPLC, and digits settings for the DC voltage function.
Example return:

```
0.02
1
dmm.DIGITS_4_5
```

Also see

[dmm.measure.setattribute\(\)](#) (on page 14-299)

dmm.measure.inputimpedance

This attribute determines when the 10 MΩ input divider is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.IMPEDANCE_10M

Usage

```
setting = dmm.measure.inputimpedance
dmm.measure.inputimpedance = setting
setting = dmm.measure.getattribute(function, dmm.ATTR_MEAS_INPUT_IMPEDANCE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_INPUT_IMPEDANCE, setting)
channel.setdmm("channellist", dmm.ATTR_MEAS_INPUT_IMPEDANCE, setting)
setting = channel.getdmm("channellist", dmm.ATTR_MEAS_INPUT_IMPEDANCE)
```

<i>setting</i>	10 MΩ for all ranges: dmm.IMPEDANCE_10M Automatic: dmm.IMPEDANCE_AUTO
<i>function</i>	See Functions
<i>channellist</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Automatic input impedance provides the lowest measure noise with the highest isolation on the device under test (DUT). When automatic input impedance is selected, the 100 mV to 10 V voltage ranges have more than 10 GΩ input impedance. For the 100 V and 1000 V ranges, a 10 MΩ input divider is placed across the HI and LO input terminals.

When the input impedance is set to 10 MΩ, the 100 mV to 1000 V ranges have a 10 MΩ input divider across the HI and LO input terminals. The 10 MΩ impedance provides stable measurements when the terminals are open (approximately 100 μV at 1 PLC).

Choosing automatic input impedance is a balance between achieving low DC voltage noise on the 100 mV and 1 V ranges and optimizing measurement noise due to charge injection. The DAQ6510 is optimized for low noise and charge injection when the DUT has less than 100 kΩ input resistance. When the DUT input impedance is more than 100 kΩ, selecting an input impedance of 10 MΩ optimizes the measurement for lowest noise on the 100 mV and 1 V ranges. You can achieve short-term low noise and low charge injection on the 100 mV and 1 V ranges with autozero off. For the 10 V to 1000 V ranges, both input impedance settings achieve low charge injection.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE  
dmm.measure.inputimpedance = dmm.IMPEDANCE_AUTO
```

Set input impedance to be set automatically when the DC voltage function is selected.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)  
channel.setdmm("101:109", dmm.ATTR_MEAS_INPUT_IMPEDANCE, dmm.IMPEDANCE_AUTO)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.
Set input impedance to be set automatically.

Also see

[channel.getdmm](#) (on page 14-75)
[channel.setdmm](#) (on page 14-88)
[dmm.digitize.inputimpedance](#) (on page 14-157)
[dmm.measure.opendetector](#) (on page 14-273)

dmm.measure.limit[Y].audible

This attribute determines if the instrument beeper sounds when a limit test passes or fails.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Continuity: dmm.AUDIBLE_PASS Other functions: dmm.AUDIBLE_NONE

Usage

```
state = dmm.measure.limit[Y].audible
dmm.measure.limit[Y].audible = state
state = dmm.measure.getattribute(function, dmm.ATTR_MEAS_LIMIT_AUDIBLE_Y)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_LIMIT_AUDIBLE_Y, state)
channel.setdmm("channelList", dmm.ATTR_MEAS_LIMIT_AUDIBLE_Y, state)
state = channel.getdmm("channelList", dmm.ATTR_MEAS_LIMIT_AUDIBLE_Y)
```

<i>state</i>	When the beeper sounds: ■ Never: dmm.AUDIBLE_NONE ■ On test failure: dmm.AUDIBLE_FAIL ■ On test pass: dmm.AUDIBLE_PASS
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	The measure function; see Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The tone and length of beeper cannot be adjusted.

Example

See [dmm.measure.limit\[Y\].low.value](#) (on page 14-252) for an example of how to use this command.

Also see

[dmm.digitize.limit\[Y\].audible](#) (on page 14-158)
[dmm.measure.limit\[Y\].enable](#) (on page 14-248)

dmm.measure.limit[Y].autoclear

This attribute indicates if the test result for limit *Y* should be cleared automatically or not.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.ON

Usage

```
value = dmm.measure.limit[Y].autoclear
dmm.measure.limit[Y].autoclear = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_Y)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_Y, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_Y, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_Y)
```

<i>value</i>	The auto clear setting: - Disable: dmm.OFF - Enable: dmm.ON
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

When autoclear is set to on, limit conditions are cleared automatically after each measurement. If you are making a series of measurements, the instrument shows the limit test result of the last measurement for the pass or fail indication for the limit.

If you want to know if any of a series of measurements failed the limit, set the autoclear setting to off. When this is set to off, a failed indication is not cleared automatically. It remains set until it is cleared with the clear command.

The autoclear setting affects both the high and low limits.

Example

See [dmm.measure.limit\[Y\].low.value](#) (on page 14-252) for an example of how to use this command.

Also see

[dmm.digitize.limit\[Y\].autoclear](#) (on page 14-159)

[dmm.measure.limit\[Y\].enable](#) (on page 14-248)

dmm.measure.limit[Y].clear()

This function clears the results of the limit test defined by *Y*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.measure.limit[Y].clear()
channel.setdmm("channelList", dmm.ATTR_MEAS_LIMIT_FAIL_Y, dmm.FAIL_NONE)
```

<i>Y</i>	Limit number: 1 or 2
<i>channelList</i>	The channels to set, using standard channel naming (on page 11-6)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Use this command to clear the test results of limit *Y* when the limit auto clear option is turned off. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, turn the auto clear option on.

Example 1

See [dmm.measure.limit\[Y\].low.value](#) (on page 14-252) for an example of how to use this command.

Example 2

```
print(channel.getdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_1))
print(channel.getdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_2))

-- Clear limit 1 conditions
channel.setdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_1, dmm.FAIL_NONE)
-- Clear limit 2 conditions
channel.setdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_2, dmm.FAIL_NONE)

print(channel.getdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_1))
print(channel.getdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_2))
```

This example outputs the fail conditions for channels 101 and 102 for limits 1 and 2. It then clears the fail conditions.

Example output showing readings on channels 101 and 102 failed limit 1 low values:

```
[1]=dmm.FAIL_LOW, [2]=dmm.FAIL_LOW
[1]=dmm.FAIL_NONE, [2]=dmm.FAIL_NONE
```

Example output showing the failed conditions are cleared:

```
[1]=dmm.FAIL_NONE, [2]=dmm.FAIL_NONE
[1]=dmm.FAIL_NONE, [2]=dmm.FAIL_NONE
```

Also see

[dmm.digitize.limit\[Y\].clear\(\)](#) (on page 14-160)

[dmm.measure.limit\[Y\].autoclear](#) (on page 14-246)

dmm.measure.limit[Y].enable

This attribute enables or disables a limit test on the measurement from the selected measure function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OFF

Usage

```
state = dmm.measure.limit[Y].enable
dmm.measure.limit[Y].enable = state
state = dmm.measure.getattribute(function, dmm.ATTR_MEAS_LIMIT_ENABLE_Y)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_LIMIT_ENABLE_Y, state)
channel.setdmm("channelList", dmm.ATTR_MEAS_LIMIT_ENABLE_Y, state)
state = channel.getdmm("channelList", dmm.ATTR_MEAS_LIMIT_ENABLE_Y)
```

state	Limit <i>Y</i> testing: - Disable: dmm.OFF - Enable: dmm.ON
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command enables or disables a limit test for the selected measurement function. When this attribute is enabled, the limit *Y* testing occurs on each measurement made by the instrument. Limit *Y* testing compares the measurements to the high-limit and low-limit values. If a measurement falls outside these limits, the test fails.

Example

See [dmm.measure.limit\[Y\].low.value](#) (on page 14-252) for an example of how to use this command.

Also see

[dmm.digitize.limit\[Y\].enable](#) (on page 14-162)

[dmm.measure.limit\[Y\].autoclear](#) (on page 14-246)

[dmm.measure.limit\[Y\].clear\(\)](#) (on page 14-247)

[dmm.measure.limit\[Y\].fail](#) (on page 14-250)

[dmm.measure.limit\[Y\].high.value](#) (on page 14-251)

[dmm.measure.limit\[Y\].low.value](#) (on page 14-252)

dmm.measure.limit[Y].fail

This attribute queries the results of a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = dmm.measure.limit[Y].fail
value = channel.getdmm("channelList", dmm.ATTR_MEAS_LIMIT_FAIL_Y)
```

<i>value</i>	The results of the limit test for limit <i>Y</i>: ■ <code>dmm.FAIL_NONE</code>: Test passed; measurement under or equal to the high limit ■ <code>dmm.FAIL_HIGH</code>: Test failed; measurement exceeded high limit ■ <code>dmm.FAIL_LOW</code>: Test failed; measurement exceeded low limit ■ <code>dmm.FAIL_BOTH</code>: Test failed; measurement exceeded both limits
<i>Y</i>	Limit number: 1 or 2
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

<code>dmm.FUNC_DC_VOLTAGE</code>	<code>dmm.FUNC_RESISTANCE</code>	<code>dmm.FUNC_ACV_FREQUENCY</code>
<code>dmm.FUNC_AC_VOLTAGE</code>	<code>dmm.FUNC_4W_RESISTANCE</code>	<code>dmm.FUNC_ACV_PERIOD</code>
<code>dmm.FUNC_DC_CURRENT</code>	<code>dmm.FUNC_DIODE</code>	<code>dmm.FUNC_DCV_RATIO</code>
<code>dmm.FUNC_AC_CURRENT</code>	<code>dmm.FUNC_CAPACITANCE</code>	<code>dmm.FUNC_DIGITIZE_CURRENT</code>
<code>dmm.FUNC_TEMPERATURE</code>	<code>dmm.FUNC_CONTINUITY</code>	<code>dmm.FUNC_DIGITIZE_VOLTAGE</code>

Details

This command queries the result of a limit test for the selected measurement function.

The response message indicates if the limit test passed or how it failed (on the high or low limit).

If autoclear is set to off, reading the results of a limit test does not clear the fail indication of the test. To clear a failure, send the clear command. To automatically clear the results, set auto clear on.

If auto clear is set to on and you are making a series of measurements, the last measurement limit determines the fail indication for the limit. If auto clear is turned off, the results return a test fail if any of one of the readings failed.

To use this attribute, you must set the limit state to on.

If the readings are stored in a reading buffer, you can use the `bufferVar.statuses` command to see the results.

Example

See [dmm.measure.limit\[Y\].low.value](#) (on page 14-252) for an example of how to use this command.

Also see

[bufferVar.statuses](#) (on page 14-61)
[dmm.digitize.limit\[Y\].fail](#) (on page 14-163)
[dmm.measure.limit\[Y\].enable](#) (on page 14-248)

dmm.measure.limit[Y].high.value

This attribute specifies the upper limit for a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1 for most functions; see Details

Usage

```
highLimit = dmm.measure.limit[Y].high.value
dmm.measure.limit[Y].high.value = highLimit
highLimit = dmm.measure.getattribute(function, dmm.ATTR_MEAS_LIMIT_HIGH_Y)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_LIMIT_HIGH_Y, highLimit)
channel.setdmm("channelList", dmm.ATTR_MEAS_LIMIT_HIGH_Y, highLimit)
highLimit = channel.getdmm("channelList", dmm.ATTR_MEAS_LIMIT_HIGH_Y)
```

<i>highLimit</i>	The value of the upper limit (-1e+12 to 1e+12)
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command sets the high limit for the limit *Y* test for the selected measurement function. When limit *Y* testing is enabled, the instrument generates a fail indication when the measurement value is more than this value.

Default is 0.8 for limit 1 when the diode function is selected; 10 when the continuity function is selected. The default for limit 2 for the diode and continuity functions is 1.

Example

See [dmm.measure.limit\[Y\].low.value](#) (on page 14-252) for an example of how to use this command.

Also see

[dmm.digitize.limit\[Y\].high.value](#) (on page 14-164)
[dmm.measure.limit\[Y\].enable](#) (on page 14-248)
[dmm.measure.limit\[Y\].low.value](#) (on page 14-252)

dmm.measure.limit[Y].low.value

This attribute specifies the lower limit for a limit test.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	-1 for most functions; see Details

Usage

```
lowLimit = dmm.measure.limit[Y].low.value
dmm.measure.limit[Y].low.value = lowLimit
lowLimit = dmm.measure.getattribute(function, dmm.ATTR_MEAS_LIMIT_LOW_Y)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_LIMIT_LOW_Y, lowLimit)
channel.setdmm("channelList", dmm.ATTR_MEAS_LIMIT_LOW_Y, lowLimit)
lowLimit = channel.getdmm("channelList", dmm.ATTR_MEAS_LIMIT_LOW_Y)
```

<i>lowLimit</i>	The low limit value of limit <i>Y</i> : -1E+12 to 1E+12)
<i>Y</i>	Limit number: 1 or 2
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command sets the lower limit for the limit *Y* test for the selected measure function. When limit *Y* testing is enabled, this causes a fail indication to occur when the measurement value is less than this value.

Default is 0.3 for limit 1 when the diode function is selected. The default for limit 2 for the diode function is -1.

Example 1

This example enables limits 1 and 2 for voltage measurements. Limit 1 is checking for readings to be between 3 and 5 V, while limit 2 is checking for the readings to be between 1 and 7 V. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail is 1. Therefore, if any one of the fails is 1, analyze the reading buffer data to determine which reading failed the limits.

```

reset()
-- Set the instrument to measure voltage
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
-- Set the range to 10 V
dmm.measure.range = 10
-- Set the nplc to 0.1
dmm.measure.nplc = 0.1
-- Disable auto clearing for limit 1
dmm.measure.limit[1].autoclear = dmm.OFF
-- Set high limit on 1 to fail if reading exceeds 5 V
dmm.measure.limit[1].high.value = 5
-- Set low limit on 1 to fail if reading is less than 3 V
dmm.measure.limit[1].low.value = 3
--- Set the beeper to sound if the reading exceeds the limits for limit 1
dmm.measure.limit[1].audible = dmm.AUDIBLE_FAIL
-- Enable limit 1 checking for voltage measurements
dmm.measure.limit[1].enable = dmm.ON
-- Disable auto clearing for limit 2
dmm.measure.limit[2].autoclear = dmm.OFF
-- Set high limit on 2 to fail if reading exceeds 7 V
dmm.measure.limit[2].high.value = 7
-- Set low limit on 2 to fail if reading is less than 1 V
dmm.measure.limit[2].low.value = 1
-- Enable limit 2 checking for voltage measurements
dmm.measure.limit[2].enable = dmm.ON
-- Set the measure count to 50
dmm.measure.count = 50
-- Create a reading buffer that can store 100 readings
LimitBuffer = buffer.make(100)
-- Make 50 readings and store them in LimitBuffer
dmm.measure.read(LimitBuffer)
-- Check if any of the 50 readings were outside of the limits
print("limit 1 results = " .. dmm.measure.limit[1].fail)
print("limit 2 results = " .. dmm.measure.limit[2].fail)
-- Clear limit 1 conditions
dmm.measure.limit[1].clear()
-- Clear limit 2 conditions
dmm.measure.limit[2].clear()

```

Example output that shows all readings are within limit values (all readings between 3 V and 5 V):

```

limit 1 results = dmm.FAIL_NONE
limit 2 results = dmm.FAIL_NONE

```

Example output showing at least one reading failed limit 1 high values (a 6 V reading would cause this condition or a reading greater than 5 V but less than 7 V):

```

limit 1 results = dmm.FAIL_HIGH
limit 2 results = dmm.FAIL_NONE

```

Example output showing at least one reading failed limit 1 and 2 low values (a 0.5 V reading would cause this condition or a reading less than 1 V):

```

limit 1 results = dmm.FAIL_LOW
limit 2 results = dmm.FAIL_LOW

```

Example 2

This example enables limits 1 and 2 for voltage measurements on channels 1 and 2 of slot 1. Limit 1 is checking for readings to be between 3 and 5 V, while limit 2 is checking for the readings to be between 1 and 7 V. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail is 1. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
reset()
sampleCount = 50
scanCount = 25

-- Set channels 1 and 2 to measure DC voltage.
-- Each has a sample count of 50.
channel.setdmm("101:102", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE,
    dmm.ATTR_MEAS_COUNT, sampleCount)

-- Set the range to 10 V.
channel.setdmm("101:102", dmm.ATTR_MEAS_RANGE, 10)

-- Set up limit 1: Disable auto clearing, set the high limit
-- to fail if reading exceeds 5 V, set low limit to fail if
-- reading is less than 3 V, and enable limit checking.
channel.setdmm("101:102", dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_1, dmm.OFF,
    dmm.ATTR_MEAS_LIMIT_HIGH_1, 5, dmm.ATTR_MEAS_LIMIT_LOW_1, 3,
    dmm.ATTR_MEAS_LIMIT_ENABLE_1, dmm.ON)

-- Set up limit 2: Disable auto clearing, set the high limit
-- to fail if reading exceeds 7 V, set low limit to fail if
-- reading is less than 1 V, and enable limit checking.
channel.setdmm("101:102", dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_2, dmm.OFF,
    dmm.ATTR_MEAS_LIMIT_HIGH_2, 7, dmm.ATTR_MEAS_LIMIT_LOW_2, 1,
    dmm.ATTR_MEAS_LIMIT_ENABLE_2, dmm.ON)

-- Set the beeper to sound if the reading exceeds the limits for limit 2.
channel.setdmm("101:102", dmm.ATTR_MEAS_LIMIT_AUDIBLE_2, dmm.AUDIBLE_FAIL)

-- Set the measure count to 50.
channel.setdmm("101:102", dmm.ATTR_MEAS_COUNT, 50)

-- Create a standard reading buffer that can store 2500 readings.
LimitBuffer = buffer.make(2 * sampleCount * scanCount, buffer.STYLE_STANDARD)

-- Make 2500 readings and store them in LimitBuffer.
scan.create("101:102")
scan.buffer = LimitBuffer
LimitBuffer.clear()
scan.scancount = scanCount
scan.scaninterval = 1.0

trigger.model.initiate()
waitcomplete()

-- Check to see if any of the readings were outside of the limits.
print(channel.getdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_1))
print(channel.getdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_2))

-- Clear limit 1 conditions
channel.setdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_1, dmm.FAIL_NONE)
-- Clear limit 2 conditions
channel.setdmm("101:102", dmm.ATTR_MEAS_LIMIT_FAIL_2, dmm.FAIL_NONE)

printbuffer(1, LimitBuffer.n, LimitBuffer)
```


Example output that shows all readings are within limit values (all readings between 3 V and 5 V):

```
limit 1 results = dmm.FAIL_NONE
```

```
limit 2 results = dmm.FAIL_NONE
```

Example output showing at least one reading failed limit 1 high values (a 6 V reading would cause this condition or a reading greater than 5 V but less than 7 V):

```
limit 1 results = dmm.FAIL_HIGH
```

```
limit 2 results = dmm.FAIL_NONE
```

Example output showing at least one reading failed limit 1 and 2 low values (a 0.5 V reading would cause this condition or a reading less than 1 V):

```
limit 1 results = dmm.FAIL_LOW
```

```
limit 2 results = dmm.FAIL_LOW
```

Also see

[dmm digitize.limit\[Y\].low.value](#) (on page 14-165)

[dmm.measure.limit\[Y\].autoclear](#) (on page 14-246)

[dmm.measure.limit\[Y\].clear\(\)](#) (on page 14-247)

[dmm.measure.limit\[Y\].enable](#) (on page 14-248)

[dmm.measure.limit\[Y\].fail](#) (on page 14-250)

[dmm.measure.limit\[Y\].high.value](#) (on page 14-251)

dmm.measure.linesync

This attribute determines if line synchronization is used during the measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OFF

Usage

```
state = dmm.measure.linesync
dmm.measure.linesync = state
state = dmm.measure.getattribute(function, dmm.ATTR_MEAS_LINE_SYNC)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_LINE_SYNC, state)
state = channel.getdmm("channellist", dmm.ATTR_MEAS_LINE_SYNC)
channel.setdmm("channellist", dmm.ATTR_MEAS_LINE_SYNC, state)
```

<i>state</i>	Disable line sync: dmm.OFF Enable line sync: dmm.ON
<i>function</i>	See Functions
<i>channellist</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

When line synchronization is enabled, measurements are initiated at the first positive-going zero crossing of the power line cycle after the trigger.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_CURRENT
dmm.measure.linesync = dmm.ON
```

Set line synchronization on for DC current measurements.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_LINE_SYNC, dmm.ON)
```

For channels 1 through 9 on slot 1, set the DMM function to DC voltage.
Set line synchronization on.

Also see

[channel.getdmm](#) (on page 14-75)
[channel.setdmm](#) (on page 14-88)
[Line cycle synchronization](#) (on page 4-72)

dmm.measure.math.enable

This attribute enables or disables math operations on measurements for the selected measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OFF

Usage

```
value = dmm.measure.math.enable
dmm.measure.math.enable = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_MATH_ENABLE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_MATH_ENABLE, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_MATH_ENABLE, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_MATH_ENABLE)
```

<i>value</i>	The math enable setting: - Disable: dmm.OFF - Enable: dmm.ON
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

When this command is set to on, the math operation specified by the math format command is performed before completing a measurement.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.math.format = dmm.MATH_PERCENT
dmm.measure.count = 1
dmm.measure.math.percent = dmm.measure.read()
dmm.measure.math.enable = dmm.ON
dmm.measure.count = 5
MathBuffer = buffer.make(100)
dmm.measure.read(MathBuffer)
printbuffer(1, MathBuffer.n, MathBuffer.formattedreadings)
dmm.measure.count = 1
for x = 1, 3 do
    print(dmm.measure.read(MathBuffer))
end
```

Configure the instrument for DC voltage measurement and reset the DC voltage function to the default settings. Set math format to percent.

Acquire 1 reading to use as the relative percent value.

Take 5 readings with percent math enabled and store them in a buffer called `MathBuffer` that can store 100 readings.

Take three additional readings without using the reading buffer.

Sample output assuming no load was connected to the instrument:

```
-100.00242 %, -100.00228 %, -100.00220 %, -100.00233 %, -100.00216 %
-100.00228175
-100.0022889
-100.00210915
```

Example 2

```
-- Set channel 1 on slot 1 to use the voltage function.
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
-- Set channel 1 to use the percentage math format and enable math.
channel.setdmm("101", dmm.ATTR_MEAS_MATH_FORMAT, dmm.MATH_PERCENT,
    dmm.ATTR_MEAS_MATH_ENABLE, dmm.ON )

-- Acquire one reading to use as the relative percent value.
channel.close("101")
dmm.measure.count = 1
dmm.measure.math.percent = dmm.measure.read()

-- Create a buffer named MathBuffer that holds 100 readings.
MathBuffer = buffer.make(100)

channel.setdmm("101", dmm.ATTR_MEAS_COUNT, 5)
channel.close("101")
dmm.measure.read(MathBuffer)

printbuffer(1, MathBuffer.n, MathBuffer.formattedreadings)
```

Configure channel 101 to measure voltage.

Set math format to percent and enable the math feature.

Acquire one reading to use as the relative percent value.

Create a buffered named `MathBuffer`.

Make five readings with percent math enabled and store them in `MathBuffer`.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.enable](#) (on page 14-169)

[dmm.measure.math.format](#) (on page 14-261)

dmm.measure.math.format

This attribute specifies which math operation is performed on measurements when math operations are enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.MATH_PERCENT

Usage

```
operation = dmm.measure.math.format
dmm.measure.math.format = operation
operation = dmm.measure.getattribute(function, dmm.ATTR_MEAS_MATH_FORMAT)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_MATH_FORMAT, operation)
channel.setdmm("channelList", dmm.ATTR_MEAS_MATH_FORMAT, operation)
operation = channel.getdmm("channelList", dmm.ATTR_MEAS_MATH_FORMAT)
```

<i>operation</i>	Math operation to be performed on measurements: ■ y = mx+b: dmm.MATH_MXB ■ Percent: dmm.MATH_PERCENT ■ Reciprocal: dmm.MATH_RECIPROCAL
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This specifies which math operation is performed on measurements for the selected measurement function.

You can choose one of the following math operations:

- **y = mx+b**: Manipulate normal display readings by adjusting the m and b factors.
- **Percent**: Displays measurements as the percentage of deviation from a specified reference constant.
- **Reciprocal**: The reciprocal math operation displays measurement values as reciprocals. The displayed value is $1/X$, where X is the measurement value (if relative offset is being used, this is the measured value with relative offset applied).

Math calculations are applied to the input signal after relative offset and before limit tests.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.math.format = dmm.MATH_RECIPROCAL
dmm.measure.math.enable = dmm.ON
```

Enables the reciprocal math operation on voltage measurements.

Example 2

```
-- Set channel 1 on slot 1 to use the voltage function.
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
-- Set channel 1 to use the percentage math format and enable math.
channel.setdmm("101", dmm.ATTR_MEAS_MATH_FORMAT, dmm.MATH_PERCENT,
    dmm.ATTR_MEAS_MATH_ENABLE, dmm.ON )

-- Acquire one reading to use as the relative percent value.
channel.close("101")
dmm.measure.count = 1
dmm.measure.math.percent = dmm.measure.read()

-- Create a buffer named MathBuffer that holds 100 readings.
MathBuffer = buffer.make(100)

channel.setdmm("101", dmm.ATTR_MEAS_COUNT, 5)
channel.close("101")
dmm.measure.read(MathBuffer)

printbuffer(1, MathBuffer.n, MathBuffer.formattedreadings)
```

Configure channel 101 to measure voltage.
Set math format to percent and enable the math feature.
Acquire one reading to use as the relative percent value.
Create a buffered named `MathBuffer`.
Make five readings with percent math enabled and store them in `MathBuffer`.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)
[dmm.digitize.math.format](#) (on page 14-172)
[dmm.measure.math.enable](#) (on page 14-258)

dmm.measure.math.mxb.bfactor

This attribute specifies the offset, b, for the $y = mx + b$ operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0

Usage

```
offsetFactor = dmm.measure.math.mxb.bfactor
dmm.measure.math.mxb.bfactor = offsetFactor
offsetFactor = dmm.measure.getattribute(function, dmm.ATTR_MEAS_MATH_MXB_BF)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_MATH_MXB_BF, offsetFactor)
channel.setdmm("channelList", dmm.ATTR_MEAS_MATH_MXB_BF, offsetFactor)
offsetFactor = channel.getdmm("channelList", dmm.ATTR_MEAS_MATH_MXB_BF)
```

<i>offsetFactor</i>	The offset for the $y = mx + b$ operation: $-1e12$ to $+1e12$
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This attribute specifies the offset (b) for an $mx+b$ operation.

The $mx+b$ math operation lets you manipulate normal display readings (x) mathematically based on the calculation:

$$y = mx + b$$

Where:

- y is the displayed result
- m is a user-defined constant for the scale factor
- x is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- b is the user-defined constant for the offset factor

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.math.format = dmm.MATH_MXB
dmm.measure.math.mxb.mfactor = 0.80
dmm.measure.math.mxb.bfactor = 50
dmm.measure.math.enable = dmm.ON
```

Set the measurement function to voltage.
Set the scale factor for the mx +b operation to 0.80.
Set the offset factor to 50.
Enable the math function.

Example 2

```
-- Set channel 1 on slot 1 to use the voltage function.
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
-- Set channel 1 to use the MXB math format, set up the factors, and enable math.
channel.setdmm("101", dmm.ATTR_MEAS_MATH_FORMAT, dmm.MATH_MXB,
    dmm.ATTR_MEAS_MATH_MXB_MF, 0.8, dmm.ATTR_MEAS_MATH_MXB_BF, 42,
    dmm.ATTR_MEAS_MATH_ENABLE, dmm.ON)
```

Enables the $y = mx + b$ math operation on digitize voltage measurements made on channel 1 of slot 1. Set the m (scale) factor to 0.8 and the b (offset) factor to 42.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.mxb.bfactor](#) (on page 14-174)

[dmm.measure.math.enable](#) (on page 14-258)

[dmm.measure.math.format](#) (on page 14-261)

[dmm.measure.math.mxb.mfactor](#) (on page 14-265)

dmm.measure.math.mxb.mfactor

This attribute specifies the scale factor, m , for the $y = mx + b$ math operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1

Usage

```
scaleFactor = dmm.measure.math.mxb.mfactor
dmm.measure.math.mxb.mfactor = scaleFactor
scaleFactor = dmm.measure.getattribute(function, dmm.ATTR_MEAS_MATH_MXB_MF)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_MATH_MXB_MF, scaleFactor)
channel.setdmm("channelList", dmm.ATTR_MEAS_MATH_MXB_MF, scaleFactor)
scaleFactor = channel.getdmm("channelList", dmm.ATTR_MEAS_MATH_MXB_MF)
```

<i>scaleFactor</i>	The scale factor: $-1e12$ to $+1e12$
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command sets the scale factor (m) for an $mx + b$ operation for the selected measurement function.

The $mx + b$ math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$$y = mx + b$$

Where:

- y is the displayed result
- m is a user-defined constant for the scale factor
- x is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- b is the user-defined constant for the offset factor

Example

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.math.format = dmm.MATH_MXB
dmm.measure.math.mxb.mfactor = 0.80
dmm.measure.math.mxb.bfactor = 50
dmm.measure.math.enable = dmm.ON
```

Set the measurement function to voltage.
Set the scale factor for the mx +b operation to 0.80.
Set the offset factor to 50.
Enable the math function.

Example 2

```
-- Set channel 1 on slot 1 to use the voltage function.
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
-- Set channel 1 to use the MXB math format, set up the factors, and enable math.
channel.setdmm("101", dmm.ATTR_MEAS_MATH_FORMAT, dmm.MATH_MXB,
    dmm.ATTR_MEAS_MATH_MXB_MF, 0.8, dmm.ATTR_MEAS_MATH_MXB_BF, 42,
    dmm.ATTR_MEAS_MATH_ENABLE, dmm.ON)
```

Enables the $y = mx + b$ math operation on digitize voltage measurements made on channel 1 of slot 1. Set the m (scale) factor to 0.8 and the b (offset) factor to 42.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.mxb.mfactor](#) (on page 14-176)

[dmm.measure.math.enable](#) (on page 14-258)

[dmm.measure.math.format](#) (on page 14-261)

[dmm.measure.math.mxb.bfactor](#) (on page 14-263)

dmm.measure.math.percent

This attribute specifies the reference constant that is used when math operations are set to percent.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1

Usage

```
reference = dmm.measure.math.percent
dmm.measure.math.percent = reference
reference = dmm.measure.getattribute(function, dmm.ATTR_MEAS_MATH_PERCENT)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_MATH_PERCENT, reference)
channel.setdmm("channelList", dmm.ATTR_MEAS_MATH_PERCENT, reference)
reference = channel.getdmm("channelList", dmm.ATTR_MEAS_MATH_PERCENT)
```

<i>reference</i>	The reference used when the math operation is set to percent: -1e12 to +1e12
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The percent math function displays measurements as percent deviation from a specified reference constant. The percent calculation is:

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

Where:

- *Percent* is the result
- *Input* is the measurement (if relative offset is being used, this is the relative offset value)
- *Reference* is the user-specified constant

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.math.format = dmm.MATH_PERCENT
dmm.measure.math.percent = 50
dmm.measure.math.enable = dmm.ON
```

Set the measurement function to voltage.
Set the math operations to percent.
Set the reference constant to 50 for voltage measurements.
Enable math operations.

Example 2

```
-- Set channel 1 on slot 1 to use the voltage function.
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
-- Set channel 1 to use the percentage math format, set the percentage,
-- and enable math.
channel.setdmm("101", dmm.ATTR_MEAS_MATH_FORMAT, dmm.MATH_PERCENT,
    dmm.ATTR_MEAS_MATH_PERCENT, 42, dmm.ATTR_MEAS_MATH_ENABLE, dmm.ON)
```

Enables the $y = mx + b$ math operation on voltage measurements made on channel 1 of slot 1. Set the m (scale) factor to 0.8 and the b (offset) factor to 42.

Also see

[Calculations that you can apply to measurements](#) (on page 4-59)

[dmm.digitize.math.percent](#) (on page 14-177)

[dmm.measure.math.enable](#) (on page 14-258)

[dmm.measure.math.format](#) (on page 14-261)

dmm.measure.nplc

This command sets the time that the input signal is measured for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1

Usage

```
nplc = dmm.measure.nplc
dmm.measure.nplc = nplc
nplc = dmm.measure.getattribute(function, dmm.ATTR_MEAS_NPLC)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_NPLC, nplc)
channel.setdmm("channelList", dmm.ATTR_MEAS_NPLC, nplc)
nplc = channel.getdmm("channelList", dmm.ATTR_MEAS_NPLC)
```

<i>nplc</i>	The number of power line cycles: 60 Hz: 0.0005 to 15 50 Hz: 0.0005 to 12
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command sets the amount of time that the input signal is measured.

The amount of time is specified as the number of power line cycles (NPLCs). Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz or 400 Hz is 20 ms (1/50). For 60 Hz, if you set the NPLC to 0.1, the measure time is 1.667 ms.

The shortest amount of time results in the fastest reading rate but increases the reading noise and decreases the number of usable digits.

The longest amount of time provides the lowest reading noise and more usable digits but has the slowest reading rate.

Settings between the fastest and slowest number of power line cycles are a compromise between speed and noise.

If you change the PLCs, you may want to adjust the displayed digits to reflect the change in usable digits.

NOTE

The measurement time can also be set as an aperture time. Changing the NPLC value changes the aperture time and changing the aperture time changes the NPLC value.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.nplc = 0.5
```

Sets the measurement function to DC voltage. Set the NPLC value to 0.5.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 0.5)
channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_5_5)
```

For channels 1 through 9, set the DMM function to DC voltage.

Set the range to 5 V, which selects the 10 V range.

Set NPLC to 0.5.

Set the number of display digits to 5½.

Also see

[channel.getdmm](#) (on page 14-75)

[channel.setdmm](#) (on page 14-88)

[dmm.measure.aperture](#) (on page 14-203)

[dmm.measure.linesync](#) (on page 14-257)

[Using aperture or NPLCs to adjust speed and accuracy](#) (on page 4-73)

dmm.measure.offsetcompensation.enable

This attribute determines if offset compensation is used.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OCOMP_AUTO

Usage

```
state = dmm.measure.offsetcompensation.enable
dmm.measure.offsetcompensation.enable = state
state = dmm.measure.getattribute(function, dmm.ATTR_MEAS_OFFCOMP_ENABLE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_OFFCOMP_ENABLE, state)
channel.setdmm("channelList", dmm.ATTR_MEAS_OFFCOMP_ENABLE, state)
state = channel.getdmm("channelList", dmm.ATTR_MEAS_OFFCOMP_ENABLE)
```

<i>state</i>	Set offset compensation to: ■ Disabled: dmm.OCOMP_OFF ■ Enabled: dmm.OCOMP_ON (for 4-wire resistance, not available for ranges more than 10 kΩ) ■ Be set on or off automatically based on other instrument settings: dmm.OCOMP_AUTO
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

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

For 4-wire resistance measurements, when offset compensation is enabled, the measure range is limited to a maximum of 10 kΩ. When Auto is selected, the instrument automatically turns offset compensation on or off as appropriate for the selected range.

For 2-wire resistance measurements, offset compensation is always set to off.

For temperature measurements, offset compensation is only available when the transducer type is set to an RTD option.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_FOURRTD
dmm.measure.offsetcompensation.enable = dmm.OCOMP_ON
print(dmm.measure.read())
```

Sets the measurement function to resistance. Set the instrument for 4-wire RTD and turn offset compensation on.

Make a measurement.

Example 2

```
channel.setdmm("101:104", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:104", dmm.ATTR_MEAS_TRANSducer, dmm.TRANS_FOURRTD)
channel.setdmm("101:104", dmm.ATTR_MEAS_OFFCOMP_ENABLE, dmm.OCOMP_OFF)
channel.setdmm("101:104", dmm.ATTR_MEAS_FOUR_RTD, dmm.RTD_PT3916)
```

For channels 1 through 4, set the DMM function to temperature.

Set the transducer type to 3-wire RTD. Set the offset compensation off. Set the RTD type to PT3916.

Also see

None

dmm.measure.opendetector

This attribute determines if the detection of open leads is enabled or disabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	4W Res: dmm.OFF Temperature: dmm.ON

Usage

```
state = dmm.measure.opendetector
dmm.measure.opendetector = state
state = dmm.measure.getattribute(function, dmm.ATTR_MEAS_OPEN_DETECTOR)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_OPEN_DETECTOR, state)
channel.setdmm("channelList", dmm.ATTR_MEAS_OPEN_DETECTOR, state)
state = channel.getdmm("channelList", dmm.ATTR_MEAS_OPEN_DETECTOR)
```

<i>state</i>	Disable open lead detector: dmm.OFF Enable open lead detector: dmm.ON
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

For temperature measurements, this is only available when the transducer is set to a thermocouple or one of the RTDs.

Long lengths of thermocouple wire can have a large amount of capacitance, which is seen at the input of the DMM. If an intermittent open occurs in the thermocouple circuit, the capacitance can cause an erroneous on-scale reading. The open thermocouple detection circuit, when enabled, applies a 100 μ A pulse of current to the thermocouple before the start of each temperature measurement.

Example 1

<pre>dmm.measure.func = dmm.FUNC_TEMPERATURE dmm.measure.transducer = dmm.TRANS_THERMOCOUPLE dmm.measure.opendetector = dmm.OFF</pre>	Set the measure function to temperature. Set the transducer type to thermocouple. Set open lead detection off.

Example 2

```
channel.setdmm("101:104", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:104", dmm.ATTR_MEAS_TRANSDUCER, dmm.TRANS_THERMOCOUPLE)
channel.setdmm("101:104", dmm.ATTR_MEAS_OPEN_DETECTOR, dmm.OFF)
```

For channels 1 through 4, set the measure function to temperature.
Set the transducer type to thermocouple. Set open lead detection off.

Also see

[dmm.measure.transducer](#) (on page 14-320)

[Open lead detection](#) (on page 4-17)

dmm.measure.range

This attribute determines the positive full-scale measure range.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	See Details

Usage

```
rangeValue = dmm.measure.range
dmm.measure.range = rangeValue
rangeValue = dmm.measure.getattribute(function, dmm.ATTR_MEAS_RANGE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_RANGE, rangeValue)
channel.setdmm("channelList", dmm.ATTR_MEAS_RANGE, rangeValue)
rangeValue = channel.getdmm("channelList", dmm.ATTR_MEAS_RANGE)
```

<i>rangeValue</i>	See Details
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

You can assign any real number using this command. The instrument selects the closest fixed range that is large enough to measure the entered number. For example, for current measurements, if you expect a reading of approximately 9 mA, set the range to 9 mA to select the 10 mA range. When you read this setting, you see the positive full-scale value of the measurement range that the instrument is presently using.

This command is primarily intended to eliminate the time that is required by the instrument to automatically search for a range.

When a range is fixed, any signal greater than the entered range generates an overrange condition. When an overrange condition occurs, the front panel displays "Overflow" and the remote interface returns 9.9e+37.

NOTE

When you set a value for the measurement range, the measurement autorange setting is automatically disabled for the selected measurement function (if supported by that function).

The range for measure functions defaults to autorange for all measure functions. If you switch from a fixed range to autorange, autorange is set to off. The range remains at the fixed range until a measurement is made, at which time the range is set to accommodate the new measurement.

The following table lists the ranges for each function.

If the measurement function is...	The available ranges are...
DC voltage	100 mV, 1 V, 10 V, 100 V, 1000 V
AC voltage	100 mV, 1 V, 10 V, 100 V, 750 V
DC current	10 μ A, 100 μ A, 1 mA, 10 mA, 100 mA, 1 A, 3 A
AC current	100 μ A, 1 mA, 10 mA, 100 mA, 1 A, 3 A
2-wire resistance	10 Ω , 100 Ω , 1 k Ω , 10 k Ω , 100 k Ω , 1 M Ω , 10 M Ω , 100 M Ω
4-wire resistance with offset compensation off	1 Ω , 10 Ω , 100 Ω , 1 k Ω , 10 k Ω , 100 k Ω , 1 M Ω , 10 M Ω , 100 M Ω
4-wire resistance with offset compensation on	1 Ω , 10 Ω , 100 Ω , 1 k Ω , 10 k Ω
Continuity	1 k Ω (fixed)
Diode	10 V (fixed)
Capacitance	1 nF, 10 nF, 100 nF, 1 μ F, 10 μ F, 100 μ F
DC voltage ratio	100 mV, 1 V, 10 V, 100 V, 1000 V

Example 1

```
dmm.measure.func = dmm.FUNC_AC_VOLTAGE
dmm.measure.range = 90
print(dmm.measure.range)
```

Set the range to 90 V, which selects the 100 V range.
Output:
100

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 5)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 0.5)
channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_5_5)
```

For channels 1 through 9, set the DMM function to DC voltage.
Set the range to 5 V, which selects the 10 V range.
Set NPLC to 0.5.
Set the number of display digits to 5½.

Also see

[channel.getdmm](#) (on page 14-75)

[channel.setdmm](#) (on page 14-88)

[dmm.digitize.range](#) (on page 14-179)

[dmm.measure.autorange](#) (on page 14-209)

dmm.measure.read()

This function makes measurements, places them in a reading buffer, and returns the last reading.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = dmm.measure.read()
reading = dmm.measure.read(bufferName)
```

<i>reading</i>	The last reading of the measurement process
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer; if no buffer is defined, it defaults to <code>defbuffer1</code>

Details

This command initiates measurements using the present function setting, stores the readings in a reading buffer, and returns the last reading.

The `dmm.measure.count` attribute determines how many measurements are made.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

Example 1

```
voltMeasBuffer = buffer.make(10000)
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
print(dmm.measure.read(voltMeasBuffer))
```

Create a buffer named `voltMeasBuffer`. Set the instrument to measure voltage. Make a measurement that is stored in the `voltMeasBuffer` and is also printed.

Example 2

```
reset()
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.close("101")
print(dmm.measure.read())
```

Set up channel 101 to measure DC voltage.
Close channel 101.
Make a measurement on the closed channel.

Also see

[buffer.make\(\)](#) (on page 14-22)
[dmm.digitize.read\(\)](#) (on page 14-180)
[dmm.measure.count](#) (on page 14-221)
[Reading buffers](#) (on page 6-1, on page 6-1)
[trigger.model.load\(\) — SimpleLoop](#) (on page 14-487)

dmm.measure.readwithtime()

This function initiates measurements and returns the last actual measurement and time information in UTC format without using the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading, seconds, fractional = dmm.measure.readwithtime()  
dmm.measure.readwithtime(bufferName)
```

<i>reading</i>	The last reading of the measurement process
<i>seconds</i>	Seconds in UTC format
<i>fractional</i>	Fractional seconds
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer; if no buffer is specified, this parameter defaults to <code>defbuffer1</code>

Details

This command initiates measurements using the present function setting, stores the readings in a reading buffer, and returns the last reading.

The `dmm.measure.count` attribute determines how many measurements are performed.

When you use a reading buffer with a command or action that makes multiple readings, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

If you define a specific reading buffer, the reading buffer must exist before you make the measurement.

Example

```
print(dmm.measure.readwithtime(defbuffer1))  
Print the last measurement and time information from defbuffer1 in UTC format, which will look similar to:  
-1.405293589829e-11 1400904629 0.1950935
```

Also see

- [dmm.digitize.readwithtime\(\)](#) (on page 14-181)
- [dmm.measure.count](#) (on page 14-221)
- [trigger.model.load\(\) — SimpleLoop](#) (on page 14-487)

dmm.measure.refjunction

This attribute selects the type of the thermocouple reference junction.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.REFJUNCT_SIMULATED

Usage

```

type = dmm.measure.refjunction
dmm.measure.refjunction = type
type = dmm.measure.getattribute(function, dmm.ATTR_MEAS_REF_JUNCTION)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_REF_JUNCTION, type)
channel.setdmm("channelList", dmm.ATTR_MEAS_REF_JUNCTION, type)
type = channel.getdmm("channelList", dmm.ATTR_MEAS_REF_JUNCTION)

```

<i>type</i>	Type of reference junction: ■ dmm.REFJUNCT_SIMULATED ■ dmm.REFJUNCT_INTERNAL ■ dmm.REFJUNCT_EXTERNAL
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Only available when the temperature function is selected and the transducer type is set to thermocouple.

When you are making rear terminal measurements, you can select the internal or external option. When the internal option is selected, the DAQ6510 periodically updates the temperature. When the external option is selected, the temperature is updated when the external reference function channel is scanned.

When you are making front terminal measurements, the only option is simulated. You can set the simulated reference temperature with the command `dmm.measure.simreftemperature`.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THERMOCOUPLE
dmm.measure.unit = dmm.UNIT_CELSIUS
dmm.measure.simreftemperature = 30
```

Sets 30 degrees Celsius as the simulated reference temperature for thermocouples.

Example 2

```
channel.setdmm("101:104", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:104", dmm.ATTR_MEAS_TRANSducer, dmm.TRANS_THERMOCOUPLE)
channel.setdmm("101:104", dmm.ATTR_MEAS_REF_JUNCTION, dmm.REFJUNCT_INTERNAL)
```

Sets channels 1 to 4 of slot 1 to use thermocouples with an internal reference temperature.

Also see

[dmm.measure.transducer](#) (on page 14-320)

[dmm.measure.simreftemperature](#) (on page 14-309)

[Temperature measurements](#) (on page 4-26)

dmm.measure.rel.acquire()

This function acquires a measurement and stores it as the relative offset value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

dmm.measure.rel.acquire()

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command triggers the instrument to make a new measurement for the selected function. This measurement is then stored as the new relative offset level.

When you send this command, the instrument does not apply any math, limit test, or filter settings to the measurement, even if they are set. It is a measurement that is made as if these settings are disabled.

If an error event occurs during the measurement, `nil` is returned and the relative offset level remains at the last valid setting.

You must change to the function for which you want to acquire a value before sending this command.

The instrument must have relative offset enabled to use the acquired relative offset value.

After executing this command, you can use the `dmm.measure.rel.level` attribute to see the last relative level value that was acquired or that was set.

Example

<pre>dmm.measure.func = dmm.FUNC_DC_VOLTAGE rel_value = dmm.measure.rel.acquire() dmm.measure.rel.enable = dmm.ON print(rel_value)</pre>	Acquires a relative offset level value for voltage measurements, turns the relative offset feature on, and outputs the value.

Also see

- [dmm.digitize.rel.acquire\(\)](#) (on page 14-183)
- [dmm.measure.rel.enable](#) (on page 14-283)
- [dmm.measure.rel.level](#) (on page 14-284)

dmm.measure.rel.enable

This attribute enables or disables the application of a relative offset value to the measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.OFF

Usage

```
state = dmm.measure.rel.enable
dmm.measure.rel.enable = state
state = dmm.measure.getattribute(function, dmm.ATTR_MEAS_REL_ENABLE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_REL_ENABLE, state)
channel.setdmm("channelList", dmm.ATTR_MEAS_REL_ENABLE, state)
state = channel.getdmm("channelList", dmm.ATTR_MEAS_REL_ENABLE)
```

<i>state</i>	The setting: - Enable: dmm.ON - Disable: dmm.OFF
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value. You can enter a relative offset value or have the instrument acquire a relative offset value.

Each returned measured relative reading is the result of the following calculation:

$$\text{Displayed reading} = \text{Actual measured reading} - \text{Relative offset value}$$

Example 1

```
dmm.measure.func = dmm.FUNC_AC_CURRENT
dmm.measure.rel.acquire()
dmm.measure.rel.enable = dmm.ON
```

Enables the relative measurements for AC current and uses the acquire command to set the relative level attribute.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_REL_ENABLE, dmm.ON)
```

For channels 1 through 9, set the DMM function to DC voltage. Enable the relative measurements.

Also see

[dmm.digitize.rel.enable](#) (on page 14-184)
[dmm.measure.rel.acquire\(\)](#) (on page 14-282)
[dmm.measure.rel.level](#) (on page 14-284)
[dmm.measure.rel.method](#) (on page 14-287)

dmm.measure.rel.level

This attribute contains the relative offset value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0

Usage

```
value = dmm.measure.rel.level
dmm.measure.rel.level = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_REL_LEVEL)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_REL_LEVEL, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_REL_LEVEL, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_REL_LEVEL)
```

<i>value</i>	Relative offset value for measurements; see Details
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command specifies the relative offset value that can be applied to new measurements. When relative offset is enabled, all subsequent measured readings are offset by the value that is set for this command.

You can set this value, or have the instrument acquire a value. If the instrument acquires the value, read this setting to return the value that was measured internally.

The ranges for the relative offset values for all functions are listed in the following table.

	Minimum	Maximum
DC voltage	-1000	1000
AC voltage	-750	750
DC current	-3	3
AC current	-3	3
Resistance	-1e+8	1e+8
4-wire resistance	-1e+8	1e+8
Diode	-10	10
Capacitance	-0.001	0.001
Temperature	-3310	3310
Continuity	-1000	1000
Frequency	-1e+6	1e+6
Period	-1	1
DC voltage ratio - Method set to result	-1e+12	1e+12
DC voltage ratio - Method set to parts	-1000	1000
Digitize voltage	-1000	1000
Digitize current	-3	3

NOTE

If you have math, limits, or filter operations selected, you can set the relative offset value to include the adjustments made by these operations. To include these operations, set `dmm.measure.rel.level` to `dmm.measure.read()`. The adjustments from these operations are not used if you use the `dmm.measure.rel.acquire()` function to set the relative offset level.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_CURRENT
dmm.measure.rel.level = dmm.measure.read()
dmm.measure.rel.enable = dmm.ON
```

Set the measure function to DC current.
Set the relative offset level to be the reading with any calculations included.
Enable the relative offset.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_REL_ENABLE, dmm.ON)
channel.setdmm("101:109", dmm.ATTR_MEAS_REL_LEVEL, 1)
```

For channels 1 through 9, set the DMM function to DC voltage. Enable the relative measurements and set the offset level to 1 V.

Also see

[Relative offset](#) (on page 4-56)

[dmm digitize.rel.level](#) (on page 14-185)

[dmm.measure.rel.acquire\(\)](#) (on page 14-282)

[dmm.measure.rel.enable](#) (on page 14-283)

dmm.measure.rel.method

This attribute determines if relative offset is applied to the measurements before calculating the DC voltage ratio value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.METHOD_PARTS

Usage

```
value = dmm.measure.rel.method
dmm.measure.rel.method = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_REL_METHOD)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_REL_METHOD, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_REL_METHOD, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_REL_METHOD)
```

<i>value</i>	The method used: - Do not apply relative offset: dmm.METHOD_RESULT - Apply relative offset: dmm.METHOD_PARTS
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command determines if relative offset is applied to the voltage measurements before the ratio calculation or if the relative offset is applied to the final calculated value.

When the parts method is selected, the individual readings each have the relative offset value applied before being used to calculate the measurement reading. The relative offset value is working with smaller ranges, so an error may occur. Reduce the relative offset value if you receive an error. When a relative offset value is acquired when the parts method is selected, the relative offset levels are made and applied to both input and sense.

A relative offset is applied to the sense value and then to the input value.

When the results method is selected, the individual readings do not have the relative offset value applied. The relative offset value is applied to the final calculation.

Example 1

```
dmm.measure.func = dmm.FUNC_DCV_RATIO
dmm.measure.rel.method = dmm.METHOD_PARTS
```

Set the measure function to DC voltage ratio.
Set the method to apply relative offset before generating the ratio.

Example 2

```
channel.setdmm("101", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DCV_RATIO)
channel.setdmm("101", dmm.ATTR_MEAS_REL_METHOD, dmm.METHOD_RESULT)
```

Set the channel measure function to DC voltage ratio.

Set the method to not apply relative offset before generating the ratio.

Also see

[dmm.measure.rel.enable](#) (on page 14-283)

[dmm.measure.rel.level](#) (on page 14-284)

[Relative offset](#) (on page 4-56)

dmm.measure.rtdalpha

This attribute contains the alpha value of a user-defined RTD.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0.00385055

Usage

```
value = dmm.measure.rtdalpha
dmm.measure.rtdalpha = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_RTD_ALPHA)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_RTD_ALPHA, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_RTD_ALPHA, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_RTD_ALPHA)
```

<i>value</i>	The RTD alpha value: 0 to 0.01
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This attribute is only valid when:

- The function is set to temperature.
- The transducer type is set to one of the RTD options.
- The RTD type is set to user-defined.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THREERTD
dmm.measure.threertd = dmm.RTD_USER
dmm.measure.rtdalpha = 0.00385
```

Set the measure function to temperature.
Set the transducer type to 3-wire RTD.
Set the RTD type to User.
Set the alpha RTD value to 0.00385.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSDUCER, dmm.TRANS_THREERTD)
channel.setdmm("101:105", dmm.ATTR_MEAS_THREE_RTD, dmm.RTD_USER)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_ALPHA, 0.00385)
```

For channels 1 through 5, set the DMM function to temperature.

Set the transducer type to 3-wire RTD.

Set the RTD type to User.

Set the alpha RTD value to 0.00385.

Also see

[dmm.measure.fourrtd](#) (on page 14-238)

[dmm.measure.threertd](#) (on page 14-315)

[dmm.measure.twortd](#) (on page 14-322)

[dmm.measure.transducer](#) (on page 14-320)

dmm.measure.rtdbeta

This attribute contains the beta value of a user-defined RTD.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0.10863

Usage

```
value = dmm.measure.rtdbeta
dmm.measure.rtdbeta = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_RTD_BETA)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_RTD_BETA, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_RTD_BETA, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_RTD_BETA)
```

<i>value</i>	The RTD beta value: 0 to 1
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This attribute is only valid when:

- The function is set to temperature.
- The transducer type is set to one of the RTD options.
- The RTD type is set to user-defined.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THREERTD
dmm.measure.threertd = dmm.RTD_USER
dmm.measure.rtdalpha = 0.00385
dmm.measure.rtdbeta = 0.10863
```

Set the measure function to temperature.
Set the transducer type to 3-wire RTD.
Set the RTD type to User.
Set the alpha RTD value to 0.00385.
Set the beta RTD value to 0.10863.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSDUCER, dmm.TRANS_THREERTD)
channel.setdmm("101:105", dmm.ATTR_MEAS_THREE_RTD, dmm.RTD_USER)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_ALPHA, 0.00385)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_BETA, 0.10863)
```

For channels 1 through 5, set the DMM function to temperature.

Set the transducer type to 3-wire RTD.

Set the RTD type to User.

Set the alpha RTD value to 0.00385.

Set the beta RTD value to 0.10863.

Also see

[dmm.measure.fourrtd](#) (on page 14-238)

[dmm.measure.threertd](#) (on page 14-315)

[dmm.measure.twortd](#) (on page 14-322)

[dmm.measure.transducer](#) (on page 14-320)

dmm.measure.rtddelta

This attribute contains the delta value of a user-defined RTD.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	1.4999

Usage

```
value = dmm.measure.rtddelta
dmm.measure.rtddelta = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_RTD_DELTA)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_RTD_DELTA, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_RTD_DELTA, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_RTD_DELTA)
```

value	The RTD delta value: 0 to 5
function	See Functions
channelList	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This attribute is only valid when:

- The function is set to temperature.
- The transducer type is set to one of the RTD options.
- The RTD type is set to user-defined.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THREERTD
dmm.measure.threertd = dmm.RTD_USER
dmm.measure.rtddelta = 1.49990
```

Set the measure function to temperature.
Set the transducer type to 3-wire RTD.
Set the RTD type to User.
Set the delta RTD value to 1.49990.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSDUCER, dmm.TRANS_THREERTD)
channel.setdmm("101:105", dmm.ATTR_MEAS_THREE_RTD, dmm.RTD_USER)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_ALPHA, 0.00385)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_BETA, 0.10863)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_DELTA, 1.49990)
```

For channels 1 through 5, set the DMM function to temperature.

Set the transducer type to 3-wire RTD.

Set the RTD type to User.

Set the alpha RTD value to 0.00385.

Set the beta RTD value to 0.10863.

Set the delta RTD value to 1.49990.

Also see

[dmm.measure.fourrtd](#) (on page 14-238)

[dmm.measure.threertd](#) (on page 14-315)

[dmm.measure.twortd](#) (on page 14-322)

[dmm.measure.transducer](#) (on page 14-320)

dmm.measure.rtdzero

This attribute contains the zero value of a user-defined RTD.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	100

Usage

```
value = dmm.measure.rtdzero
dmm.measure.rtdzero = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_RTD_ZERO)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_RTD_ZERO, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_RTD_ZERO, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_RTD_ZERO)
```

<i>value</i>	The zero value of the RTD: 0 to 10000
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This attribute is only valid when:

- The function is set to temperature.
- The transducer type is set to one of the RTD options.
- The RTD type is set to user-defined.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THREERTD
dmm.measure.threertd = dmm.RTD_USER
dmm.measure.rtdalpha = 0.00385
dmm.measure.rtdzero = 120
```

Set the measure function to temperature.
Set the transducer type to 3-wire RTD.
Set the RTD type to User.
Set the alpha RTD value to 0.00385.
Set the zero RTD value to 120.

Example 2


```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSDUCER, dmm.TRANS_THREERTD)
channel.setdmm("101:105", dmm.ATTR_MEAS_THREE_RTD, dmm.RTD_USER)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_ALPHA, 0.00385)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_BETA, 0.10863)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_DELTA, 1.49990)
channel.setdmm("101:105", dmm.ATTR_MEAS_RTD_ZERO, 120)
```

For channels 1 through 10, set the DMM function to temperature.

Set the transducer type to 3-wire RTD.

Set the RTD type to User.

Set the alpha RTD value to 0.00385.

Set the beta RTD value to 0.10863.

Set the delta RTD value to 1.49990.

Set the zero RTD value to 120.

Also see

[dmm.measure.fourrtid](#) (on page 14-238)

[dmm.measure.threertid](#) (on page 14-315)

[dmm.measure.twortid](#) (on page 14-322)

[dmm.measure.transducer](#) (on page 14-320)

dmm.measure.sense.autorange

This attribute returns the setting of sense autorange.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	dmm.OFF

Usage

```

setting = dmm.measure.sense.autorange
setting = channel.getdmm("channelList", dmm.ATTR_MEAS_SENSE_RANGE_AUTO)

```

setting	Autorange disabled: dmm.OFF
channelList	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This attribute returns the setting of the sense autorange function.

Example

```

dmm.measure.func = dmm.FUNC_DCV_RATIO
print(dmm.measure.sense.autorange)

```

Select the DC voltage ratio function.
Output the setting of sense autorange. Example output:
dmm.OFF

Also see

- [Ranges](#) (on page 4-54)
- [dmm.measure.sense.range](#) (on page 14-298)

dmm.measure.sense.range

This attribute displays the positive full-scale range that is being used for the sense measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = dmm.measure.sense.range
value = channel.getdmm("channelList", dmm.ATTR_MEAS_SENSE_RANGE)
```

<i>value</i>	Range in volts
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

Displays the full-scale input that is used for the reference measurement in the denominator of the ratio.

Example 1

```
dmm.measure.func = dmm.FUNC_DCV_RATIO
print(dmm.measure.sense.range)
```

Select the DC voltage ratio function.
Output the sense range. Example output:
10

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DCV_RATIO)
print(channel.getdmm("101:105", dmm.ATTR_MEAS_SENSE_RANGE))
```

For channels 1 through 5, set the DMM function to DC voltage ratio.
Set the transducer type to 3-wire RTD. Set the RTD type to D100.
Example output:
[1]=10, [2]=10, [3]=10, [4]=10, [5]=10, [6]=10, [7]=10, [8]=10, [9]=10, [10]=10

Also see

[Ranges](#) (on page 4-54)
[dmm.measure.sense.autorange](#) (on page 14-297)

dmm.measure.setattribute()

This function allows you to set up a measure function whether or not the function is active.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Not applicable

Usage

```
dmm.measure.setattribute(function, setting, value)
```

<i>function</i>	The function for which you are setting parameters; see Functions
<i>setting</i>	The parameter for the function; refer to Details and the tables following the examples
<i>value</i>	The parameter value

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The lists that follow the example and "also see" listings show the parameters that are available for each function. Each parameter includes a link to the descriptions of the corresponding TSP command descriptions. The options for each parameter are the same as the settings for the TSP commands.

Example

```
dmm.measure.setattribute(dmm.FUNC_DC_CURRENT, dmm.ATTR_MEAS_RANGE, 35e-6)
dmm.measure.setattribute(dmm.FUNC_DC_CURRENT, dmm.ATTR_MEAS_DIGITS,
    dmm.DIGITS_5_5)
dmm.measure.setattribute(dmm.FUNC_DC_CURRENT, dmm.ATTR_MEAS_NPLC, 0.5)
dmm.measure.configlist.create("measlist")
dmm.measure.configlist.storefunc("measlist", dmm.FUNC_DC_CURRENT)
```

Configure the DC Current function settings for the range, display resolution, and number of power line cycles (NPLCs) whether or not DC Current is the active function.
Create a configuration list named `measlist`.
Save the settings into `measlist`.

Also see

[dmm.measure.configlist.storefunc\(\)](#) (on page 14-220)

[dmm.measure.getattribute\(\)](#) (on page 14-242)

DC voltage (dmm.FUNC_DC_VOLTAGE)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE
[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO
[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO
[dB reference](#) (on page 14-223): dmm.ATTR_MEAS_DB_REFERENCE
[dBm reference](#) (on page 14-225): dmm.ATTR_MEAS_DBM_REFERENCE
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Input impedance](#) (on page 14-243): dmm.ATTR_MEAS_INPUT_IMPEDANCE
[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC
[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE
[Unit](#) (on page 14-324): dmm.ATTR_MEAS_UNIT
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_N

AC voltage (dmm.FUNC_AC_VOLTAGE)

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO
[dB reference](#) (on page 14-223): dmm.ATTR_MEAS_DB_REFERENCE
[dBm reference](#) (on page 14-225): dmm.ATTR_MEAS_DBM_REFERENCE
[Detector bandwidth](#) (on page 14-227): dmm.ATTR_MEAS_DETECTBW
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE
[Unit](#) (on page 14-324): dmm.ATTR_MEAS_UNIT
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_N

DC current (dmm.FUNC_DC_CURRENT)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO

[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC

[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE

[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ *N*

AC current (dmm.FUNC_AC_CURRENT)

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO

[Detector bandwidth](#) (on page 14-227): dmm.ATTR_MEAS_DETECTBW

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE

[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ *N*

Temperature (dmm.FUNC_TEMPERATURE)

[2-wire RTD type](#) (on page 14-322): dmm.ATTR_MEAS_TWO_RTD

[3-wire RTD type](#) (on page 14-315): dmm.ATTR_MEAS_THREE_RTD

[4-wire RTD type](#) (on page 14-238): dmm.ATTR_MEAS_FOUR_RTD

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC

[Offset Compensation](#) (on page 14-271): dmm.ATTR_MEAS_OFFCOMP_ENABLE

[Open lead detector](#) (on page 14-273): dmm.ATTR_MEAS_OPEN_DETECTOR

[Reference junction](#) (on page 14-280): dmm.ATTR_MEAS_REF_JUNCTION

[RTD Alpha](#) (on page 14-289): dmm.ATTR_MEAS_RTD_ALPHA

[RTD Beta](#) (on page 14-291): dmm.ATTR_MEAS_RTD_BETA

[RTD Delta](#) (on page 14-293): dmm.ATTR_MEAS_RTD_DELTA

[RTD Zero](#) (on page 14-295): dmm.ATTR_MEAS_RTD_ZERO

[Simulated reference temperature](#) (on page 14-309): dmm.ATTR_MEAS_SIM_REF_TEMP

[Thermistor](#) (on page 14-311): dmm.ATTR_MEAS_THERMISTOR

[Thermocouple](#) (on page 14-313): dmm.ATTR_MEAS_THERMOCOUPLE

[Transducer](#) (on page 14-320): dmm.ATTR_MEAS_TRANSDUCER

[Unit](#) (on page 14-324): dmm.ATTR_MEAS_UNIT

[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ *N*

2-wire resistance (dmm.FUNC_RESISTANCE)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO

[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC

[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE

[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_*N*

4-wire resistance (dmm.FUNC_4W_RESISTANCE)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO

[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC

[Offset compensation](#) (on page 14-271): dmm.ATTR_MEAS_OFFCOMP_ENABLE

[Open lead detector](#) (on page 14-273): dmm.ATTR_MEAS_OPEN_DETECTOR

[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE

[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_*N*

Diode (dmm.FUNC_DIODE)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE
[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO
[Bias level](#) (on page 14-212): dmm.ATTR_MEAS_BIAS_LEVEL
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Capacitance (dmm.FUNC_CAPACITANCE)

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Continuity (dmm.FUNC_CONTINUITY)

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Frequency (dmm.FUNC_ACV_FREQUENCY)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE
[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY
[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS
[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT
[Threshold autorange](#) (on page 14-317): dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO
[Threshold range](#) (on page 14-319): dmm.ATTR_MEAS_THRESHOLD_RANGE
[User delay](#) (on page 14-326) *N* (where *N* is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_

Period (dmm.FUNC_ACV_PERIOD)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[Threshold autorange](#) (on page 14-317): dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO

[Threshold range](#) (on page 14-319): dmm.ATTR_MEAS_THRESHOLD_RANGE

[User delay](#) (on page 14-326) N (where N is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ N

DCV ratio (dmm.FUNC_DCV_RATIO)

[Aperture](#) (on page 14-203): dmm.ATTR_MEAS_APERTURE

[Auto delay](#) (on page 14-205): dmm.ATTR_MEAS_AUTO_DELAY

[Auto range](#) (on page 14-209): dmm.ATTR_MEAS_RANGE_AUTO

[Auto zero](#) (on page 14-207): dmm.ATTR_MEAS_AUTO_ZERO

[Display digits](#) (on page 14-228): dmm.ATTR_MEAS_DIGITS

[Line sync](#) (on page 14-257): dmm.ATTR_MEAS_LINE_SYNC

[Measure count](#) (on page 14-221): dmm.ATTR_MEAS_COUNT

[NPLC](#) (on page 14-269): dmm.ATTR_MEAS_NPLC

[Range](#) (on page 14-275): dmm.ATTR_MEAS_RANGE

[Sense range \(read only\)](#) (on page 14-298): dmm.ATTR_MEAS_SENSE_RANGE

[User delay](#) (on page 14-326) N (where N is 1 to 5): dmm.ATTR_MEAS_USER_DELAY_ N

Digitize current (dmm.FUNC_DIGITIZE_CURRENT)

[Aperture](#) (on page 14-148): dmm.ATTR_DIGI_APERTURE

[Count](#) (on page 14-150): dmm.ATTR_DIGI_COUNT

[Display digits](#) (on page 14-154): dmm.ATTR_DIGI_DIGITS

[Range](#) (on page 14-179): dmm.ATTR_DIGI_RANGE

[Sample rate](#) (on page 14-187): dmm.ATTR_DIGI_SAMPLE_RATE

[Unit](#) (on page 14-189): dmm.ATTR_DIGI_UNIT

[User delay](#) (on page 14-190) N (where N is 1 to 5): dmm.ATTR_DIGI_USER_DELAY_ N

Digitize voltage (dmm.FUNC_DIGITIZE_VOLTAGE)

[Aperture](#) (on page 14-148): dmm.ATTR_DIGI_APERTURE

[Count](#) (on page 14-150): dmm.ATTR_DIGI_COUNT

[Decibel reference](#) (on page 14-152): dmm.ATTR_DIGI_DB_REFERENCE

[Decibel-milliwatts reference](#) (on page 14-153): dmm.ATTR_DIGI_DBM_REFERENCE

[Display digits](#) (on page 14-154): dmm.ATTR_DIGI_DIGITS

[Input impedance](#) (on page 14-157): dmm.ATTR_DIGI_INPUT_IMPEDANCE

[Range](#) (on page 14-179): dmm.ATTR_DIGI_RANGE

[Relative enable](#) (on page 14-184): dmm.ATTR_DIGI_REL_ENABLE

[Relative level](#) (on page 14-185): dmm.ATTR_DIGI_REL_LEVEL

[Sample rate](#) (on page 14-187): dmm.ATTR_DIGI_SAMPLE_RATE

[Unit](#) (on page 14-189): dmm.ATTR_DIGI_UNIT

[User delay](#) (on page 14-190) N (where N is 1 to 5): dmm.ATTR_DIGI_USER_DELAY_ N

Math options (measure)

[Enable math](#) (on page 14-258): dmm.ATTR_MEAS_MATH_ENABLE

[b \(offset\) value](#) (on page 14-263): dmm.ATTR_MEAS_MATH_MXB_BF

[m \(scalar\) value](#) (on page 14-265): dmm.ATTR_MEAS_MATH_MXB_MF

[Math format](#) (on page 14-261): dmm.ATTR_MEAS_MATH_FORMAT

[Percent](#) (on page 14-267): dmm.ATTR_MEAS_MATH_PERCENT

Math options (digitize)

[Enable math](#) (on page 14-169): dmm.ATTR_DIGI_MATH_ENABLE

[b \(offset\) value](#) (on page 14-174): dmm.ATTR_DIGI_MATH_MXB_BF

[m \(scalar\) value](#) (on page 14-176): dmm.ATTR_DIGI_MATH_MXB_MF

[Math format](#) (on page 14-172): dmm.ATTR_DIGI_MATH_FORMAT

[Percent](#) (on page 14-177): dmm.ATTR_DIGI_MATH_PERCENT

Limit options (measure)

[Limit 1 audible](#) (on page 14-245): `dmm.ATTR_MEAS_LIMIT_AUDIBLE_1`

[Limit 1 auto clear](#) (on page 14-246): `dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_1`

[Limit 1 enable](#) (on page 14-248): `dmm.ATTR_MEAS_LIMIT_ENABLE_1`

[Limit 1 fail](#) (on page 14-250): `dmm.ATTR_MEAS_LIMIT_FAIL_1`

[Limit 1 high value](#) (on page 14-251): `dmm.ATTR_MEAS_LIMIT_HIGH_1`

[Limit 1 low value](#) (on page 14-252): `dmm.ATTR_MEAS_LIMIT_LOW_1`

[Limit 2 audible](#) (on page 14-245): `dmm.ATTR_MEAS_LIMIT_AUDIBLE_2`

[Limit 2 auto clear](#) (on page 14-246): `dmm.ATTR_MEAS_LIMIT_AUTO_CLEAR_2`

[Limit 2 enable](#) (on page 14-248): `dmm.ATTR_MEAS_LIMIT_ENABLE_2`

[Limit 2 fail](#) (on page 14-250): `dmm.ATTR_MEAS_LIMIT_FAIL_2`

[Limit 2 high value](#) (on page 14-251): `dmm.ATTR_MEAS_LIMIT_HIGH_2`

[Limit 2 low value](#) (on page 14-252): `dmm.ATTR_MEAS_LIMIT_LOW_2`

Limit options (digitize)

[Limit 1 audible](#) (on page 14-158): `dmm.ATTR_DIGI_LIMIT_AUDIBLE_1`

[Limit 1 auto clear](#) (on page 14-159): `dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_1`

[Limit 1 enable](#) (on page 14-162): `dmm.ATTR_DIGI_LIMIT_ENABLE_1`

[Limit 1 fail](#) (on page 14-163): `dmm.ATTR_DIGI_LIMIT_FAIL_1`

[Limit 1 high value](#) (on page 14-164): `dmm.ATTR_DIGI_LIMIT_HIGH_1`

[Limit 1 low value](#) (on page 14-165): `dmm.ATTR_DIGI_LIMIT_LOW_1`

[Limit 2 audible](#) (on page 14-158): `dmm.ATTR_DIGI_LIMIT_AUDIBLE_2`

[Limit 2 auto clear](#) (on page 14-159): `dmm.ATTR_DIGI_LIMIT_AUTO_CLEAR_2`

[Limit 2 enable](#) (on page 14-162): `dmm.ATTR_DIGI_LIMIT_ENABLE_2`

[Limit 2 fail](#) (on page 14-163): `dmm.ATTR_DIGI_LIMIT_FAIL_2`

[Limit 2 high value](#) (on page 14-164): `dmm.ATTR_DIGI_LIMIT_HIGH_2`

[Limit 2 low value](#) (on page 14-165): `dmm.ATTR_DIGI_LIMIT_LOW_2`

Analog trigger settings (measurement functions)

[Edge level](#) (on page 14-191): `dmm.ATTR_MEAS_ATRIG_EDGE_LEVEL`

[Edge slope](#) (on page 14-193): `dmm.ATTR_MEAS_ATRIG_EDGE_SLOPE`

[Mode](#) (on page 14-140): `dmm.ATTR_DIGI_ATRIG_MODE`

[Window direction](#) (on page 14-197): `dmm.ATTR_MEAS_ATRIG_WINDOW_DIRECTION`

[Window level high](#) (on page 14-199): `dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_HIGH`

[Window level low](#) (on page 14-201): `dmm.ATTR_MEAS_ATRIG_WINDOW_LEVEL_LOW`

Analog trigger settings (digitize functions)

[Edge level](#) (on page 14-137): `dmm.ATTR_DIGI_ATRIG_EDGE_LEVEL`

[Edge slope](#) (on page 14-138): `dmm.ATTR_DIGI_ATRIG_EDGE_SLOPE`

[Mode](#) (on page 14-140): `dmm.ATTR_DIGI_ATRIG_MODE`

[Window direction](#) (on page 14-142): `dmm.ATTR_DIGI_ATRIG_WINDOW_DIRECTION`

[Window level high](#) (on page 14-144): `dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_HIGH`

[Window level low](#) (on page 14-146): `dmm.ATTR_DIGI_ATRIG_WINDOW_LEVEL_LOW`

Filter options (measure only)

[Filter enable](#) (on page 14-232): `dmm.ATTR_MEAS_FILTER_ENABLE`

[Filter count](#) (on page 14-230): `dmm.ATTR_MEAS_FILTER_COUNT`

[Filter type](#) (on page 14-234): `dmm.ATTR_MEAS_FILTER_TYPE`

[Filter window](#) (on page 14-236): `dmm.ATTR_MEAS_FILTER_WINDOW`

Relative offset settings (measurement functions)

[Relative offset enable](#) (on page 14-283): `dmm.ATTR_MEAS_REL_ENABLE`

[Relative offset method](#) (on page 14-287) (DCV ratio measurements only):
`dmm.ATTR_MEAS_REL_METHOD`

[Relative offset value](#) (on page 14-284): `dmm.ATTR_MEAS_REL_LEVEL`

Relative offset settings (digitize functions)

[Relative offset enable](#) (on page 14-184): `dmm.ATTR_DIGI_REL_ENABLE`

[Relative offset value](#) (on page 14-185): `dmm.ATTR_DIGI_REL_LEVEL`

NOTE

When you are using relative offset on a closed channel, make sure to set the relative offset level on that channel before enabling the offset.

dmm.measure.simreftemperature

This attribute sets the simulated reference temperature of the thermocouple reference junction.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Celsius: 23 Kelvin: 296.15 Fahrenheit: 73.4

Usage

```

value = dmm.measure.simreftemperature
dmm.measure.simreftemperature = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_SIM_REF_TEMP)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_SIM_REF_TEMP, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_SIM_REF_TEMP, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_SIM_REF_TEMP)

```

<i>value</i>	The simulated reference temperature: ■ Celsius: 0 to 65 ■ Kelvin: 273.15 to 338.15 ■ Fahrenheit: 32 to 149
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This attribute applies to the temperature function when the transducer type is set to thermocouple and the reference junction is set to simulated. It allows you to set the simulated reference temperature value.

Example 1

```

dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THERMOCOUPLE
dmm.measure.unit = dmm.UNIT_CELSIUS
dmm.measure.simreftemperature = 30

```

Sets 30 degrees Celsius as the simulated reference temperature for thermocouples.

Example 2

```

channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSducer, dmm.TRANS_THERMOCOUPLE)
channel.setdmm("101:105", dmm.ATTR_MEAS_THERMOCOUPLE, dmm.THERMOCOUPLE_J,
dmm.ATTR_MEAS_SIM_REF_TEMP, 30)

```

For channels 1 through 5, set the measure function to temperature.
Set the transducer type to thermocouple. Set the thermocouple type to J with a simulated reference temperature of 30.

Also see

[dmm.measure.transducer](#) (on page 14-320)

[Reference junctions](#) (on page 4-88)

[Temperature measurements](#) (on page 4-26)

dmm.measure.thermistor

This attribute describes the type of thermistor.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.THERM_5000

Usage

```
value = dmm.measure.thermistor
dmm.measure.thermistor = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_THERMISTOR)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_THERMISTOR, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_THERMISTOR, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_THERMISTOR)
```

<i>value</i>	The thermistor type in ohms: 2252 Ω: dmm.THERM_2252 5000 Ω: dmm.THERM_5000 10000 Ω: dmm.THERM_10000
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command is only applicable when the transducer type is set to thermistor.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THERMISTOR
dmm.measure.thermistor = dmm.THERM_2252
```

Set measurement function to temperature.
Set the transducer type to thermistor.
Set the thermistor type to 2252.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSducer, dmm.TRANS_THERMISTOR)
channel.setdmm("101:105", dmm.ATTR_MEAS_THERMISTOR, dmm.THERM_2252)
```

For channels 1 through 5, set the DMM function to temperature.
Set the transducer type to 3-wire RTD. Set the RTD type to User. Set the alpha RTD value to 0.00385.

Also see

[dmm.measure.transducer](#) (on page 14-320)

[Temperature measurements](#) (on page 4-26)

dmm.measure.thermocouple

This attribute indicates the thermocouple type.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.THERMOCOUPLE_K

Usage

```

value = dmm.measure.thermocouple
dmm.measure.thermocouple = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_THERMOCOUPLE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_THERMOCOUPLE, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_THERMOCOUPLE, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_THERMOCOUPLE)

```

<i>value</i>	The thermocouple type: ■ dmm.THERMOCOUPLE_B ■ dmm.THERMOCOUPLE_E ■ dmm.THERMOCOUPLE_J ■ dmm.THERMOCOUPLE_K ■ dmm.THERMOCOUPLE_N ■ dmm.THERMOCOUPLE_R ■ dmm.THERMOCOUPLE_S ■ dmm.THERMOCOUPLE_T
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command is only applicable when the transducer type is set to thermocouple.

Example 1

```

dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THERMOCOUPLE
dmm.measure.thermocouple = dmm.THERMOCOUPLE_J

```

Set the measure function to temperature.
Set the transducer type to thermocouple.
Set the thermocouple type to J.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSducer, dmm.TRANS_THERMOCOUPLE)
channel.setdmm("101:105", dmm.ATTR_MEAS_THERMOCOUPLE, dmm.THERMOCOUPLE_J)
```

For channels 1 through 5, set the measure function to temperature.
Set the transducer type to thermocouple. Set the thermocouple type to J.

Also see

[dmm.measure.transducer](#) (on page 14-320)

[dmm.measure.simreftemperature](#) (on page 14-309)

[Temperature measurements](#) (on page 4-26)

dmm.measure.threertd

This attribute defines the type of three-wire RTD that is being used.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.RTD_PT100

Usage

```
value = dmm.measure.threertd
dmm.measure.threertd = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_THREE_RTD)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_THREE_RTD, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_THREE_RTD, value)
value = channel.getdmm("channelList", dmm.ATTR_MEAS_THREE_RTD)
```

<i>value</i>	The type for 3-wire RTD: ■ PT100: dmm.RTD_PT100 ■ PT385: dmm.RTD_PT385 ■ PT3916: dmm.RTD_PT3916 ■ D100: dmm.RTD_D100 ■ F100: dmm.RTD_F100 ■ User-specified type: dmm.RTD_USER
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The transducer type must be set to temperature and the transducer must be set to 3-wire RTD before you can set the RTD type.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THREERTD
dmm.measure.threertd = dmm.RTD_D100
```

Set the measure function to temperature.
Set the transducer type to 3-wire RTD.
Set the RTD type to D100.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSDUCER, dmm.TRANS_THREERTD)
channel.setdmm("101:105", dmm.ATTR_MEAS_THREE_RTD, dmm.RTD_D100)
```

For channels 1 through 5, set the DMM function to temperature.
Set the transducer type to 3-wire RTD. Set the RTD type to D100.

Also see

[dmm.measure.rtdalpha](#) (on page 14-289)
[dmm.measure.rtdbeta](#) (on page 14-291)
[dmm.measure.rtddelta](#) (on page 14-293)
[dmm.measure.rtdzero](#) (on page 14-295)
[dmm.measure.transducer](#) (on page 14-320)
[Temperature measurements](#) (on page 4-26)

dmm.measure.threshold.autorange

This attribute determines if the threshold range is set manually or automatically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.ON

Usage

```
state = dmm.measure.threshold.autorange
dmm.measure.threshold.autorange = state
state = dmm.measure.getattribute(function, dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO, state)
channel.setdmm("channellist", dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO, state)
state = channel.getdmm("channellist", dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO)
```

<i>state</i>	The autorange setting: - Disable: dmm.OFF - Enable: dmm.ON
<i>function</i>	See Functions

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

This command determines how the range is selected.

When this command is set to off, you must set the range. If you do not set the range, the instrument remains at the range that was last selected by autorange.

When this command is set to on, the instrument uses the signal to determine the most sensitive range on which to perform the measurement. The instrument sets the range when a measurement is requested. To set the range, the instrument makes a measurement to determine the range before making the final measurement, which can result in slower reading times. Turn autorange off and set a specific range to increase measure time.

If a range is manually selected through the front panel or a remote command, this command is automatically set to off.

Example 1

```
dmm.measure.func = dmm.FUNC_ACV_PERIOD
dmm.measure.threshold.autorange = dmm.ON
```

Set the measure function to period.

Set the threshold autorange on.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_ACV_PERIOD)  
channel.setdmm("101:105", dmm.ATTR_MEAS_THRESHOLD_RANGE_AUTO, dmm.ON)
```

For channels 1 through 5, set the measure function to period.
Set the threshold autorange on.

Also see

[dmm.measure.threshold.range](#) (on page 14-319)

dmm.measure.threshold.range

This attribute indicates the expected input level of the voltage signal.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	10 (10 V)

Usage

```
range = dmm.measure.threshold.range
dmm.measure.threshold.range = range
range = dmm.measure.getattribute(function, dmm.ATTR_MEAS_THRESHOLD_RANGE)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_THRESHOLD_RANGE, range)
channel.setdmm("channelList", dmm.ATTR_MEAS_THRESHOLD_RANGE, range)
range = channel.getdmm("channelList", dmm.ATTR_MEAS_THRESHOLD_RANGE)
```

<i>range</i>	The range: 0.1 to 750; instrument selects nearest valid range (100 mV, 1 V, 10 V, 100 V, 750 V)
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The range setting conditions the signal. The instrument automatically selects the most sensitive threshold range for the value you enter. For example, if you specify the expected input voltage to be 90 mV, the instrument automatically selects the 100 mV threshold range.

Example 1

```
dmm.measure.func = dmm.FUNC_ACV_PERIOD
dmm.measure.threshold.range = 50
```

Set the threshold range for the selected function to the nearest range of 100 V.

Example 2

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_PERIOD)
channel.setdmm("101:109", dmm.ATTR_MEAS_THRESHOLD_RANGE, 50)
```

For channels 1 through 9, set the measure function to period.
Set the threshold range to 50 V, which selects the 100 V range.

Also see

[dmm.measure.threshold.autorange](#) (on page 14-317)

dmm.measure.transducer

This attribute sets the transducer type for the temperature measurement function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.TRANS_THERMOCOUPLE

Usage

```
type = dmm.measure.transducer
dmm.measure.transducer = type
type = dmm.measure.getattribute(function, dmm.ATTR_MEAS_TRANSDUCER)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_TRANSDUCER, type)
channel.setdmm("channelList", dmm.ATTR_MEAS_TRANSDUCER, type)
type = channel.getdmm("channelList", dmm.ATTR_MEAS_TRANSDUCER)
```

<i>type</i>	The transducer type: ■ Thermocouple: dmm.TRANS_THERMOCOUPLE ■ Thermistor: dmm.TRANS_THERMISTOR ■ 2-wire RTD: dmm.TRANS_TWORTD ■ 3-wire RTD: dmm.TRANS_THREERTD ■ 4-wire RTD: dmm.TRANS_FOURRTD
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The transducer type determines the type of temperature measurement that is made. Each transducer type has related settings that must also be set. For example, thermocouple measurements are only made if the type is set is set to thermocouple. You also need to set the thermocouple type when setting up a thermocouple. For the RTD transducer types, you also set the RTD type.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_THREERTD
dmm.measure.threertd = dmm.RTD_D100
```

Set the measure function to temperature.
Set the transducer type to 3-wire RTD.
Set the RTD type to D100.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSducer, dmm.TRANS_THREERTD)
channel.setdmm("101:105", dmm.ATTR_MEAS_THREE_RTD, dmm.RTD_D100)
```

For channels 1 through 5, set the DMM function to temperature.
Set the transducer type to 3-wire RTD.
Set the RTD type to D100.

Also see

[dmm.measure.fourrtid](#) (on page 14-238)
[dmm.measure.thermistor](#) (on page 14-311)
[dmm.measure.thermocouple](#) (on page 14-313)
[dmm.measure.threertd](#) (on page 14-315)
[dmm.measure.twortd](#) (on page 14-322)
[Temperature measurements](#) (on page 4-26)

dmm.measure.twordt

This attribute defines the type of 2-wire RTD that is being used.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	dmm.RTD_PT100

Usage

```
type = dmm.measure.twordt
dmm.measure.twordt = type
type = dmm.measure.getattribute(function, dmm.ATTR_MEAS_TWO_RTD)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_TWO_RTD, type)
channel.setdmm("channelList", dmm.ATTR_MEAS_TWO_RTD, type)
type = channel.getdmm("channelList", dmm.ATTR_MEAS_TWO_RTD)
```

<i>type</i>	The type of 2-wire RTD: ■ PT100: dmm.RTD_PT100 ■ PT385: dmm.RTD_PT385 ■ PT3916: dmm.RTD_PT3916 ■ D100: dmm.RTD_D100 ■ F100: dmm.RTD_F100 ■ User-specified type: dmm.RTD_USER
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The transducer type must be set to temperature and the transducer must be set to 2-wire RTD before you can set the RTD type.

Example 1

```
dmm.measure.func = dmm.FUNC_TEMPERATURE
dmm.measure.transducer = dmm.TRANS_TWORTD
dmm.measure.twordt = dmm.RTD_D100
```

Set the measure function to temperature.
Set the transducer type to 2-wire RTD.
Set the RTD type to D100.

Example 2

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
channel.setdmm("101:105", dmm.ATTR_MEAS_TRANSDUCER, dmm.TRANS_TWORTD)
channel.setdmm("101:105", dmm.ATTR_MEAS_TWO_RTD, dmm.RTD_D100)
```

For channels 1 through 5, set the DMM function to temperature.
Set the transducer type to 2-wire RTD. Set the RTD type to D100.

Also see

[dmm.measure.rtdalpha](#) (on page 14-289)

[dmm.measure.rtdbeta](#) (on page 14-291)

[dmm.measure.rtddelta](#) (on page 14-293)

[dmm.measure.rtdzero](#) (on page 14-295)

[dmm.measure.transducer](#) (on page 14-320)

[Temperature measurements](#) (on page 4-26)

dmm.measure.unit

This attribute sets the units of measurement that are displayed on the front panel of the instrument and stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	Temperature: dmm.UNIT_CELSIUS Voltage: dmm.UNIT_VOLT

Usage

```
value = dmm.measure.unit
dmm.measure.unit = value
value = dmm.measure.getattribute(function, dmm.ATTR_MEAS_UNIT)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_UNIT, value)
channel.setdmm("channelList", dmm.ATTR_MEAS_UNIT, value)
channel.getdmm("channelList", dmm.ATTR_MEAS_UNIT)
```

<i>value</i>	For DC volts and AC volts, select from the following units: ■ dmm.UNIT_VOLT ■ dmm.UNIT_DB ■ dmm.UNIT_DBM For temperature, select from the following units: ■ dmm.UNIT_CELSIUS ■ dmm.UNIT_KELVIN ■ dmm.UNIT_FAHRENHEIT
<i>function</i>	See Functions
<i>channelList</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

The change in measurement units is displayed when the next measurement is made. You can only change the units for the listed functions.

Example 1

```
dmm.measure.func = dmm.FUNC_DC_VOLTAGE  
dmm.measure.unit = dmm.UNIT_DB
```

Changes the front-panel display and buffer readings for voltage measurements to be displayed as decibel readings.

Example 2

```
channel.setdmm("201:210", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)  
channel.setdmm("201:210", dmm.ATTR_MEAS_UNIT, dmm.UNIT_DB)  
channel.setdmm("201:210", dmm.ATTR_MEAS_DB_REFERENCE, 5)
```

For channels 1 through 10 on slot 2, set the DMM function to DC voltage.
Set the units to decibels.
Set the dB reference to 5 V.

Also see

[channel.getdmm](#) (on page 14-75)
[channel.setdmm](#) (on page 14-88)
[dmm.digitize.unit](#) (on page 14-189)

dmm.measure.userdelay[N]

This attribute sets a user-defined delay that you can use in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Measure configuration list	Configuration script Measure configuration list	0 (0 s)

Usage

```

delayTime = dmm.measure.userdelay[N]
dmm.measure.userdelay[N] = delayTime
delayTime = dmm.measure.getattribute(function, dmm.ATTR_MEAS_USER_DELAY_N)
dmm.measure.setattribute(function, dmm.ATTR_MEAS_USER_DELAY_N, delayTime)
channel.setdmm("channellist", dmm.ATTR_MEAS_USER_DELAY_N, delayTime)
channel.getdmm("channellist", dmm.ATTR_MEAS_USER_DELAY_N)

```

<i>delayTime</i>	The delay: 0 for no delay, or 167 ns to 10 ks
<i>N</i>	The user delay to which this time applies: 1 to 5
<i>function</i>	The measurement function; see Functions
<i>channellist</i>	The channels to set, using standard channel naming (on page 14-3)

Functions

dmm.FUNC_DC_VOLTAGE	dmm.FUNC_RESISTANCE	dmm.FUNC_ACV_FREQUENCY
dmm.FUNC_AC_VOLTAGE	dmm.FUNC_4W_RESISTANCE	dmm.FUNC_ACV_PERIOD
dmm.FUNC_DC_CURRENT	dmm.FUNC_DIODE	dmm.FUNC_DCV_RATIO
dmm.FUNC_AC_CURRENT	dmm.FUNC_CAPACITANCE	dmm.FUNC_DIGITIZE_CURRENT
dmm.FUNC_TEMPERATURE	dmm.FUNC_CONTINUITY	dmm.FUNC_DIGITIZE_VOLTAGE

Details

To use this command in a trigger model, assign the delay to the dynamic delay block.

The delay is specific to the selected function.

Example

```

trigger.model.load("Empty")
dmm.measure.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M1)
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(3, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()

```

Set user delay 1 for measurements to 5 s.
Set trigger block 1 to a dynamic delay that calls user delay 1.
Set trigger block 2 to make or digitize a measurement.
Set trigger block 3 to branch to block 1 ten times.
Start the trigger model.

Also see

[dmm.digitize.userdelay\[N\]](#) (on page 14-190)

[trigger.model.setblock\(\)](#) — [trigger.BLOCK_DELAY_DYNAMIC](#) (on page 14-508)

dmm.reset()

This function resets commands that begin with `dmm.` to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.reset()
```

Example

<code>dmm.reset()</code>	Resets the DMM commands to their default settings.

Also see

[reset\(\)](#) (on page 14-373)

dmm.terminals

This attribute describes which set of input and output terminals the instrument is using.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
terminals = dmm.terminals
```

<code>terminals</code>	Using the front-panel input and output terminals: <code>dmm.TERMINALS_FRONT</code> Using the rear-panel input and output terminals: <code>dmm.TERMINALS_REAR</code>

Details

You must use the front-panel TERMINALS switch to change which set of terminals the instrument reads.

Example

<code>print(dmm.terminals)</code>	Request information on which terminals are used. Output if the instrument is using the front terminals: <code>dmm.TERMINALS_FRONT</code>

Also see

None

eventlog.clear()

This function clears the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.clear()
```

Details

This command removes all events from the event log, including entries in the front-panel event log.

Also see

[eventlog.next\(\)](#) (on page 14-330)
[eventlog.save\(\)](#) (on page 14-333)
[Using the event log](#) (on page 3-80)

eventlog.getcount()

This function returns the number of unread events in the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.getcount()  
eventlog.getcount(eventType)
```

eventType

Limits the return to specific event log types; set a cumulative integer value that represents the event log types to:

- Errors only: `eventlog.SEV_ERROR` or 1
- Warnings only: `eventlog.SEV_WARN` or 2
- Errors and warnings only: `eventlog.SEV_WARN|eventlog.SEV_ERROR` or 3
- Information only: `eventlog.SEV_INFO` or 4
- Errors and information only: `eventlog.SEV_INFO|eventlog.SEV_ERROR` or 5
- Warnings and information only:
`eventlog.SEV_INFO|eventlog.SEV_WARN` or 6
- All events: `eventlog.SEV_ALL` or 7

Details

A count finds the number of unread events in the event log. You can specify the event types to return or return the count for all events.

This command reports the number of events that have occurred since the command was last sent or since the event log was last cleared.

Events are read automatically when `localnode.showevents` is enabled. You can also read them individually with `eventlog.next()`.

Example

```
print(eventlog.getcount(eventlog.SEV_INFO))
```

Displays the present number of unread information messages in the instrument event log.

If there are three information messages in the event log, output is:

```
3
```

Also see

[eventlog.clear\(\)](#) (on page 14-328)

[eventlog.next\(\)](#) (on page 14-330)

[localnode.showevents](#) (on page 14-362)

[Using the event log](#) (on page 3-80)

eventlog.next()

This function returns the oldest unread event message from the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventNumber, message, severity, nodeID, timeSeconds, timeNanoSeconds =  
    eventlog.next()  
eventNumber, message, severity, nodeID, timeSeconds, timeNanoSeconds =  
    eventlog.next(eventType)
```

<i>eventNumber</i>	The event number
<i>message</i>	A description of the event
<i>severity</i>	The severity of the event: ▪ Error: 1 ▪ Warning: 2 ▪ Information: 4
<i>nodeID</i>	The TSP-Link node where the event occurred or 0 if the event occurred on the local node
<i>timeSeconds</i>	The seconds portion of the time when the event occurred
<i>timeNanoSeconds</i>	The fractional seconds portion of the time when the event occurred
<i>eventType</i>	Limits the return to specific event log types; set a cumulative integer value that represents the event log types to: ▪ Errors only: <code>eventlog.SEV_ERROR</code> or 1 ▪ Warnings only: <code>eventlog.SEV_WARN</code> or 2 ▪ Errors and warnings only: <code>eventlog.SEV_WARN eventlog.SEV_ERROR</code> or 3 ▪ Information only: <code>eventlog.SEV_INFO</code> or 4 ▪ Errors and information only: <code>eventlog.SEV_INFO eventlog.SEV_ERROR</code> or 5 ▪ Warnings and information only: <code>eventlog.SEV_INFO eventlog.SEV_WARN</code> or 6 ▪ All events: <code>eventlog.SEV_ALL</code> or 7

Details

When an event occurs on the instrument, it is placed in the event log. The `eventlog.next()` command retrieves an unread event from the event log. Once an event is read, it can no longer be accessed remotely. However, it can be viewed on the front panel. When `localnode.showevents` is enabled, this command never returns an event because those events are automatically read and sent to the remote interface.

To read multiple events, execute this command multiple times.

If there are no entries in the event log, the following is returned:

```
0    No error      0      0      0      0
```

If the event type is not defined, an event of any type is returned.

Example

```
print(eventlog.next(5))
```

Get the oldest error or information event from the event log.

Example output:

```
-285 TSP Syntax error at line 1: unexpected symbol near `0' 1 0 1367806152
652040060
```

Also see

[eventlog.clear\(\)](#) (on page 14-328)

[eventlog.getcount\(\)](#) (on page 14-328)

[eventlog.save\(\)](#) (on page 14-333)

[Using the event log](#) (on page 3-80)

eventlog.post()

This function allows you to post your own text to the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.post("message")
eventlog.post("message", eventType)
```

<i>message</i>	String that contains the message
<i>eventType</i>	The type of event; if no event is defined, defaults to <code>eventlog.SEV_INFO</code> : ■ Error: <code>eventlog.SEV_ERROR</code> or 1 ■ Warning: <code>eventlog.SEV_WARN</code> or 2 ■ Information: <code>eventlog.SEV_INFO</code> or 4

Details

You can use this command to create your own event log entries and assign a severity level to them. This can be useful for debugging and status reporting.

From the front panel, you must set the Log Warnings and Log Information options on to have the custom warning and information events placed into the event log.

You can use the following codes to create special characters in the message.

Code	Special character (description)
\018	Ω (ohm)
\019	° (degree symbol)
\020	μ (mu)
\021	Thin space (less than i)
\178	² (superscript 2, for example, x ²)
\179	³ (superscript 3, for example, x ³)
\185	Δ (delta)
\188	1/x (reciprocal)
\189	Ratio

These characters are displayed on the front-panel display only.

Example

```
eventlog.clear()
eventlog.post("Results in \018", eventlog.SEV_ERROR)
print(eventlog.next())
```

Posts an event that states "Results in Ω ".

Output:

1005 User: Results in Ω

Also see

[Using the event log](#) (on page 3-80)

eventlog.save()

This function saves the event log to a file on a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.save("filename")
eventlog.save("filename", eventType)
```

filename	A string that represents the name of the file to be saved
eventType	Limits the return to specific event log types; set a cumulative integer value that represents the event log types to: ▪ Errors only: eventlog.SEV_ERROR or 1 ▪ Warnings only: eventlog.SEV_WARN or 2 ▪ Errors and warnings only: eventlog.SEV_WARN eventlog.SEV_ERROR or 3 ▪ Information only: eventlog.SEV_INFO or 4 ▪ Errors and information only: eventlog.SEV_INFO eventlog.SEV_ERROR or 5 ▪ Warnings and information only: eventlog.SEV_INFO eventlog.SEV_WARN or 6 ▪ All events: eventlog.SEV_ALL or 7 (default)

Details

This command saves all event log entries to a USB flash drive.

If you do not define an event type, the instrument saves all event log entries.

The extension `.csv` is automatically added to the file name.

Example

eventlog.save("/usb1/WarningsApril", eventlog.SEV_WARN)	Save warning messages to a .csv file on a USB flash drive.

Also see

[eventlog.next\(\)](#) (on page 14-330)

exit()

This function stops a script that is presently running.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
exit()
```

Details

Terminates script execution when called from a script that is being executed.

This command does not wait for overlapped commands to complete before terminating script execution. If overlapped commands are required to finish, use the `waitcomplete()` function before calling `exit()`.

Also see

[waitcomplete\(\)](#) (on page 14-572)

file.close()

This function closes a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.close(fileName)
```

<i>fileName</i>

The file number returned from the <code>file.open()</code> function to close

Details

Note that files are automatically closed when the file descriptors are garbage collected.

Example

```
file_num = file.open("/usb1/GENTRIGGER", file.MODE_WRITE)
file.close(file_num)
```

Open the file GENTRIGGER for writing, then close it.

Also see

[file.open\(\)](#) (on page 14-337)

file.flush()

This function writes buffering data to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.flush(fileNumber)
```

<i>fileNumber</i>	The file number returned from the <code>file.open()</code> function to flush

Details

The `file.write()` function may be buffering data instead of writing immediately to the USB flash drive. Use `file.flush()` to flush this data. Data may be lost if the file is not closed or flushed before a script ends.

If there is going to be a time delay before more data is written to a file, flush the file to prevent loss of data because of an aborted test.

Example

```
reset()

-- Fix the range to 10 V
dmm.measure.range = 10

-- Set the measurement count to 100
dmm.measure.count = 100

-- Set up reading buffers
-- Ensure the default measurement buffer size matches the count
defbuffer1.capacity = 100
dmm.measure.read()

testDir = "TestData5"

-- create a directory on the USB drive for the data
file.mkdir(testDir)
fileName = "/usb1/" .. testDir .. "/myTestData.csv"
buffer.save(defbuffer1, fileName)

if file.usbdriveexists() != 0 then
    -- testDir = "TestData3"

    -- Create a directory on the USB drive for the data
    -- file.mkdir(testDir)
    -- Open the file where the data will be stored
    -- fileName = "/usb1/" .. testDir .. "/myTestData.csv"
    fileNumber = file.open(fileName, file.MODE_APPEND)
    -- Write header separator to file
    file.write(fileNumber,
        "\n\n=====\\n")
    -- Write the string data to a file
    file.write(fileNumber, "Tested to Company Standard ABC.123\\n")
    -- Ensure a hurry-up of data written to the file before close or script end
    file.flush(fileNumber)
    -- Close the data file
    file.close(fileNumber)

end
```

This example writes a string that indicates that the readings were made for a certain reason, such as to test to a company standard.

Also see

None

file.mkdir()

This function creates a directory at the specified path on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.mkdir("path")
```

<i>path</i>	A string that contains the path of the directory

Details

The directory path must be absolute. The name of the directory must not already exist on the flash drive.

Example

<pre>file.mkdir("TestData")</pre>	Create a new directory named <code>TestData</code> .

Also see

None

file.open()

This function opens a file on the USB flash drive for later reference.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileNumber = file.open("fileName", accessType)
```

<i>fileNumber</i>	A number identifying the open file that you use with other file commands to write, read, flush, or close the file after opening
<i>fileName</i>	A string that contains the file name to open, including the full path of file
<i>accessType</i>	The type of action to do: - Append the file: <code>file.MODE_APPEND</code> - Read the file: <code>file.MODE_READ</code> - Write to the file: <code>file.MODE_WRITE</code>

Details

The path to the file to open must be absolute.

If you use `file.MODE_APPEND` and the file specified to open is not on the flash drive, the file is created.

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

```
"/usb1/"
```

Example

```
file_num = file.open("/usb1/testfile.txt", file.MODE_WRITE)
if file_num != nil then
    file.write(file_num, "This is my test file")
    file.close(file_num)
end
```

Opens file `testfile.txt` for writing. If no errors were found while opening, writes `This is my test file` and closes the file.

Also see

None

file.read()

This function reads data from a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileContents = file.read(fileNumber, readAction)
```

<i>fileContents</i>	The contents of the file based on the <i>readAction</i> parameter
<i>fileNumber</i>	The file number returned from the <code>file.open()</code> function to read
<i>readAction</i>	The action: - Return the next line; returns <code>nil</code> if the present file position is at the end of the file: <code>file.READ_LINE</code> - Return a string that represents the number found; returns an event string if no number was found; returns <code>nil</code> if the current file position is at the end of file: <code>file.READ_NUMBER</code> - Return the whole file, starting at the present position; returns <code>nil</code> if the present file position is at the end of the file: <code>file.READ_ALL</code>

Details

This command reads data from a file.

Example

```
file_num = file.open("/usb1/testfile.txt", file.MODE_READ)
if file_num != nil then
    file_contents = file.read(file_num, file.READ_ALL)
    file.close(file_num)
end
```

Open `testfile.txt` on the USB flash drive for reading. If it opens successfully, read the entire contents of the file and store it in variable `file_contents`.
Close the file.

Also see

None

file.usbdriveexists()

This function detects if a USB flash drive is inserted into the front-panel USB port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
driveInserted = file.usbdriveexists()
```

<i>driveInserted</i>	No flash drive is detected: 0 Flash drive is detected: 1

Details

You can call this command from a script to verify that a USB flash drive is inserted before attempting to write data to it.

Example

```
local response
local xMax = 10

for x = 1, xMax do
  -- Make xMax readings and store them in defbuffer1.
  dmm.measure.read()
end

if (file.usbdriveexists() == 1) then
  response = display.BUTTON_YES
else
  response = display.input.prompt(display.BUTTONS_YESNO, "Insert a USB flash
drive.\nPress Yes to write data or No to not write data.")
end

if (response == display.BUTTON_YES) then
  if (file.usbdriveexists() == 1) then
    FileNumber = file.open("/usb1/TenReadings.csv", file.MODE_WRITE)
    file.write(FileNumber, "Reading,Seconds\n")

    -- Print out the measured values in a two-column format.
    print("\nIteration:\tReading:\tTime:\n")

    for i = 1, xMax do
      print(i, defbuffer1[i], defbuffer1.relativetimestamps[i])
      file.write(FileNumber, string.format("%g, %g\r\n", defbuffer1.readings[i],
        defbuffer1.relativetimestamps[i]))
    end
    file.close(FileNumber)
  else
    response = display.input.prompt(display.BUTTONS_OK,
      "No drive detected. Allow more time after inserting a drive.")
  end
end
```

Make measurements.
Verify that a flash drive is inserted into the instrument.
If the flash drive is inserted, write the data to the flash drive.
Print data into a two-column format.
If the flash drive is not inserted after selecting *Yes*, another prompt is displayed.

Also see

None

file.write()

This function writes data to a file on the USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
file.write(fileName, "string")
```

<i>fileName</i>	The file number returned from the <code>file.open()</code> function to write
<i>string</i>	A string that contains the data to write to the file

Details

The `file.write()` function may include data that is buffering; it may not be written to the USB flash drive immediately. Use the `file.flush()` function to immediately write buffered data to the drive.
You must use the `file.close()` command to close the file after writing.

Example

<pre>file_num = file.open("testfile.txt", file.MODE_WRITE) if file_num != nil then file.write(file_num, "This is my test file") file.close(file_num) end</pre>	Opens file <code>testfile.txt</code> for writing. If no errors were found while opening, writes <code>This is my test file</code> and closes the file.

Also see

[file.close\(\)](#) (on page 14-334)
[file.flush\(\)](#) (on page 14-335)

format.asciiprecision

This attribute sets the precision (number of digits) for all numbers returned in the ASCII format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	0 (Automatic)

Usage

```
precision = format.asciiprecision
format.asciiprecision = precision
```

<i>precision</i>	A number representing the number of digits to be printed for numbers printed with the <code>print()</code> , <code>printbuffer()</code> , and <code>printnumber()</code> functions; must be a number from 1 to 16; set to 0 to have the instrument select the precision automatically based on the number that is being formatted

Details

This attribute specifies the precision (number of digits) for numeric data printed with the `print()`, `printbuffer()`, and `printnumber()` functions. The `format.asciiprecision` attribute is only used with the ASCII format. The precision value must be a number from 0 to 16.

Note that the precision is the number of significant digits printed. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

<pre>format.asciiprecision = 10 x = 2.54 printnumber(x) format.asciiprecision = 3 printnumber(x)</pre>	Output: <pre>2.540000000e+00 2.54e+00</pre>

Also see

- [format.byteorder](#) (on page 14-342)
- [format.data](#) (on page 14-343)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)
- [printnumber\(\)](#) (on page 14-372)

format.byteorder

This attribute sets the binary byte order for the data that is printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	format.LITTLEENDIAN

Usage

```
order = format.byteorder
format.byteorder = order
```

`order`

Byte order value as follows:

- Most significant byte first: `format.BIGENDIAN`
- Least significant byte first: `format.LITTLEENDIAN`

Details

This attribute selects the byte order in which data is written when you are printing data values with the `printnumber()` and `printbuffer()` functions. The byte order attribute is only used with the `format.REAL32` and `format.REAL64` data formats.

If you are sending data to a computer with a Microsoft Windows operating system, select the `format.LITTLEENDIAN` byte order.

Example

```
x = 1.23
format.data = format.REAL32
format.byteorder = format.LITTLEENDIAN
printnumber(x)
format.byteorder = format.BIGENDIAN
printnumber(x)
```

The output depends on the terminal program you use, but it looks something like:
#0πp??
#0??pπ

Also see

- [format.asciiprecision](#) (on page 14-341)
- [format.data](#) (on page 14-343)
- [printbuffer\(\)](#) (on page 14-369)
- [printnumber\(\)](#) (on page 14-372)

format.data

This attribute sets the data format for data that is printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	format.ASCII

Usage

```
value = format.data
format.data = value
```

value

The format to use for data, set to one of the following values:

- ASCII format: `format.ASCII`
- Single-precision IEEE Std 754 binary format: `format.REAL32`
- Double-precision IEEE Std 754 binary format: `format.REAL64`

Details

You can control the precision of numeric values with the `format.asciiprecision` attribute. If `format.REAL32` or `format.REAL64` is selected, you can select the byte order with the `format.byteorder` attribute.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with `#0` and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

Example

```
format.asciiprecision = 10
x = 3.14159265
format.data = format.ASCII
printnumber(x)
format.data = format.REAL64
printnumber(x)
```

Output a number represented by `x` in ASCII using a precision of 10, then output the same number in binary using double-precision format.

Output:

```
3.141592650e+00
#0ñÔÈSû!  @
```

Also see

- [format.asciiprecision](#) (on page 14-341)
- [format.byteorder](#) (on page 14-342)
- [printbuffer\(\)](#) (on page 14-369)
- [printnumber\(\)](#) (on page 14-372)

fs.chdir()

This function sets the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
workingDirectory = fs.chdir("path")
```

<i>workingDirectory</i>	Returned value containing the working path
<i>path</i>	A string indicating the new working directory path

Details

The new working directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the given path does not exist.

Example

```
if fs.is_dir("/usb1/temp") == true then
  fs.chdir("/usb1/temp")
  testPath = fs.cwd()
  print(testPath)
else
  testPath = fs.cwd()
  print(testPath)
end
```

Insert a USB flash drive into the front panel of the instrument.
Verify that /usb1/temp is a directory and change it to be the current working directory.
Set the variable for the current working directory to be testPath.
The return should be:
/usb1/temp
If /usb1/temp is not a directory, set the variable for the current working directory to be testPath.
The return is:
/usb1

Also see

None

fs.cwd()

This function returns the absolute path of the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.cwd()
```

path	The absolute path of the current working directory

Example

```
if fs.is_dir("/usb1/temp") == true then
  fs.chdir("/usb1/temp")
  testPath = fs.cwd()
  print(testPath)
else
  testPath = fs.cwd()
  print(testPath)
end
```

Insert a USB flash drive into the front panel of the instrument.
Verify that /usb1/temp is a directory and change it to be the current working directory.
Set the variable for the current working directory to be testPath.
The return should be:
/usb1/temp
If /usb1/temp is not a directory, set the variable for the current working directory to be testPath.
The return is:
/usb1

Also see

None

fs.is_dir()

This function tests whether or not the specified path refers to a directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status = fs.is_dir("path")
```

status	Whether or not the given path is a directory (true or false)
path	The path of the file system entry to test

Details

The file system path may be absolute or relative to the current working system path.

Example 1

```
print("Is directory: ", fs.is_dir("/usb1/"))
```

Because /usb1/ is always the root directory of an inserted flash drive, you can use this command to verify that USB flash drive is inserted.

Example 2

```
if fs.is_dir("/usb1/temp") == false then
    fs.mkdir("/usb1/temp")
end
```

Insert a USB flash drive into the front panel of the instrument.
Check to see if the temp directory exists.
If it does not exist, create a directory named temp.

Also see

[fs.is_file\(\)](#) (on page 14-347)

fs.is_file()

Tests whether the specified path refers to a file (as opposed to a directory).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status = fs.is_file("path")
```

status	true if the given path is a file; otherwise, false
path	The path of the file system entry to test

Details

The file system path may be absolute or relative to the current working system path.

Example

```
rootDirectory = "/usb1/"
print("Is file: ", fs.is_file(rootDirectory))
```

Insert a USB flash drive into the front panel of the instrument.
Set `rootDirectory` to be the USB port.
Check to see if `rootDirectory` is a file. Because `rootDirectory` was set up as a directory, the return is `false`.

Also see

[fs.is_dir\(\)](#) (on page 14-346)

fs.mkdir()

This function creates a directory at the specified path.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.mkdir("newPath")
```

path	The returned path of the new directory
newpath	Location (path) of where to create the new directory

Details

The directory path may be absolute or relative to the current working directory.

An error is logged to the error queue if the parent folder of the new directory does not exist, or if a file system entry already exists at the given path.

Example

```
if fs.is_dir("/usb1/temp") == false then
    fs.mkdir("/usb1/temp")
end
```

Insert a USB flash drive into the front panel of the instrument.
Check to see if the temp directory exists.
If it does not exist, create a directory named temp.

Also see

[fs.rmdir\(\)](#) (on page 14-350)

fs.readdir()

This function returns a list of the file system entries in the directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
files = fs.readdir("path")
```

files	A table containing the names of all the file system entries in the specified directory
path	The directory path

Details

The directory path may be absolute or relative to the current working directory.

This command is nonrecursive. For example, entries in subfolders are not returned.

An error is logged to the error queue if the given path does not exist or does not represent a directory.

Example

```
rootDirectory = "/usb1/"
entries = fs.readdir(rootDirectory)
count = table.getn(entries)
print("Found a total of "..count.." files and directories")
for i = 1, count do
    print(entries[i])
end
```

Insert a USB flash drive into the front panel of the instrument.
Set `rootDirectory` to be the USB port.
Set `entries` as the variable for the file system entries in `rootDirectory`.
Return the number of files and directories in the directory.

Also see

None

fs.rmdir()

This function removes a directory from the file system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
fs.rmdir("path")
```

<i>path</i>	The path of the directory to remove

Details

This path may be absolute or relative to the current working directory.

An error is logged to the error queue if the given path does not exist or does not represent a directory.
An error is also logged if the directory is not empty.

Example

```
rootDirectory = "/usb1/"
tempDirectoryName = "temp"
if fs.is_dir(rootDirectory..tempDirectoryName) == false then
    fs.mkdir(rootDirectory..tempDirectoryName)
end
fs.rmdir(rootDirectory..tempDirectoryName)
```

Insert a USB flash drive into the front panel of the instrument.
Set `rootDirectory` to be the USB port.
Set `tempDirectoryName` to be equivalent to `temp`.
Check to see if `tempDirectoryName` exists.
If it does not exist, create a directory named `temp`.
Remove the directory.

Also see

[fs.mkdir\(\)](#) (on page 14-348)

gpib.address

This attribute contains the GPIB address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Nonvolatile memory	16

Usage

```
address = gpib.address
gpib.address = address
```

<i>address</i>	The GPIB address of the instrument: 1 to 30

Details

The address can be set to any address value from 1 to 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

A new GPIB address takes effect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so use care when setting this attribute from the GPIB interface.

You should allow sufficient time for the command to be processed before attempting to communicate with the instrument again.

The `reset()` function does not affect the GPIB address.

Example

```
gpiib.address = 26
address = gpiib.address
print(address)
```

Sets the GPIB address and reads the address.
Output:
26

Also see

[GPIB setup](#) (on page 2-10)

lan.dstprotection

This command allows you to prevent LAN connections from being closed inadvertently.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	lan.OFF

Usage

```
lan.dstprotection = state
state = lan.dstprotection
```

<code>state</code>	Allow inadvertent closure of LAN connections: <code>lan.OFF</code> or 0 Prevent inadvertent closure of LAN connections: <code>lan.ON</code> or 1

Details

When DST protection is set to off, a simple open-and-close on the DST port (5030) closes any open LAN connections.

When DST protection is set to on, to close any open LAN connection, including the DST port, you must:

1. Open the DST port.
2. Enter the system login and password.
3. Close the DST port.

Setting DST protection to on prevents LAN connections from being inadvertently closed. For example, a LAN connection could be closed if your IT department performs a port scan across the corporate network.

Example

```
lan.dstprotection = lan.ON  
print(lan.dstprotection)
```

Set DST protection on.

Query to verify the setting. The response to the query should be:

1

Also see

None

lan.ipconfig()

This function specifies the LAN configuration for the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No	LXI LAN reset	Nonvolatile memory	lan.MODE_AUTO

Usage

```
method, ipV4Address, subnetMask, gateway = lan.ipconfig()  
lan.ipconfig(method)  
lan.ipconfig(method, "ipV4Address")  
lan.ipconfig(method, "ipV4Address", "subnetMask")  
lan.ipconfig(method, "ipV4Address", "subnetMask", "gateway")
```

<i>method</i>	The method for configuring LAN settings; it can be one of the following values: ■ The instrument automatically assigns LAN settings: <code>lan.MODE_AUTO</code> ■ Manually specify the LAN settings: <code>lan.MODE_MANUAL</code>
<i>ipV4Address</i>	LAN IP address; must be a string specifying the IP address in dotted decimal notation
<i>subnetMask</i>	The LAN subnet mask; must be a string in dotted decimal notation
<i>gateway</i>	The LAN default gateway; must be a string in dotted decimal notation

Details

This command specifies how the LAN IP address and other LAN settings are assigned. If automatic configuration is selected, the instrument automatically determines the LAN information. When method is automatic, the instrument first attempts to configure the LAN settings using dynamic host configuration protocol (DHCP). If DHCP fails, it tries dynamic link local addressing (DLLA). If DLLA fails, an error occurs.

If manual is selected, you must define the IP address. You can also assign a subnet mask, and default gateway. The IP address, subnet mask, and default gateway must be formatted in four groups of numbers, each separated by a decimal. If you do not specify a subnet mask or default gateway, the previous settings are used.

Example

```
lan.ipconfig(lan.MODE_AUTO)  
print(lan.ipconfig())  
lan.ipconfig(lan.MODE_MANUAL, "192.168.0.7", "255.255.240.0", "192.168.0.3")  
print(lan.ipconfig())  
  
Set the IP configuration method to automatic. Request the IP configuration. Example output:  
lan.MODE_AUTO 134.63.78.136 255.255.254.0 134.63.78.1  
Set the IP configuration method to manual. Request the IP configuration. Output:  
lan.MODE_MANUAL 192.168.0.7 255.255.240.0 192.168.0.3
```

Also see

None

lan.lxidomain

This attribute contains the LXI domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0

Usage

```
domain = lan.lxidomain  
lan.lxidomain = domain
```

<i>domain</i>	The LXI domain number (0 to 255)

Details

This attribute sets the LXI domain number.

All outgoing LXI packets are generated with this domain number. All inbound LXI packets are ignored unless they have this domain number.

Example

```
print(lan.lxidomain)  
Displays the LXI domain.
```

Also see

None

lan.macaddress

This attribute describes the LAN MAC address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	Not applicable	Not applicable	Not applicable

Usage

```
MACaddress = lan.macaddress
```

<i>MACaddress</i>	The MAC address of the instrument

Details

The MAC address is a character string representing the MAC address of the instrument in hexadecimal notation. The string includes colons that separate the address octets.

Example

<pre>print(lan.macaddress)</pre>	Returns the MAC address. For example: 08:00:11:00:00:57

Also see

[lan.ipconfig\(\)](#) (on page 14-353)

localnode.access

This attribute contains the type of access users have to the instrument through different interfaces.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	localnode.ACCESS_FULL

Usage

```
accessType = localnode.access
localnode.access = accessType
```

<i>accessType</i>	The type of access: - Full access for all users from all interfaces: <code>localnode.ACCESS_FULL</code> - Allows access by one remote interface at a time with logins required from other interfaces: <code>localnode.ACCESS_EXCLUSIVE</code> - Allows access by one remote interface at a time with passwords required on all interfaces: <code>localnode.ACCESS_PROTECTED</code> - Allows access by one interface (including the front panel) at a time with passwords required on all interfaces: <code>localnode.ACCESS_LOCKOUT</code>

Details

When access is set to full, the instrument accepts commands from any interface with no login or password.

When access is set to exclusive, you must log out of one remote interface and log into another one to change interfaces. You do not need a password with this access.

Protected access is similar to exclusive access, except that you must enter a password when logging in.

When the access is set to locked out, a password is required to change interfaces, including the front-panel interface.

Under any access type, if a script is running on one remote interface when a command comes in from another remote interface, the command is ignored and the message "FAILURE: A script is running, use ABORT to stop it" is generated.

Example

<pre>localnode.access = localnode.ACCESS_LOCKOUT login admin logout</pre>	Set the instrument access to locked out. Log into the interface using the default password. Log out of the interface.

Also see

[localnode.password](#) (on page 14-358)

localnode.gettime()

This function retrieves the instrument date and time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.gettime()
```

Details

The time is returned in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

Example

```
print(os.date('%c', gettime()))
```

Example output:

```
Tue Dec 5 03:44:37 2017
```

Also see

[localnode.settime\(\)](#) (on page 14-361)

localnode.gettimewithfractional()

This function retrieves the number of seconds elapsed since January 1, 1970, with fractional seconds appended to the returned response.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.gettimewithfractional()
```

Also see

[localnode.gettime\(\)](#) (on page 14-356)

[localnode.settime\(\)](#) (on page 14-361)

localnode.linefreq

This attribute contains the power line frequency setting that is used for NPLC calculations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle	Not applicable	Not applicable

Usage

```
frequency = localnode.linefreq
```

<code>frequency</code>	The detected line frequency: 50 or 60

Details

The instrument automatically detects the power line frequency when the instrument is powered on. Power line frequency can be 50 Hz, 60 Hz, or 400 Hz. If the line frequency is 400 Hz, 50 is returned.

Example

<pre>frequency = localnode.linefreq print(frequency)</pre>	Reads the line frequency setting.

Also see

None

localnode.model

This attribute stores the model number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
model = localnode.model
```

<code>model</code>	The model number of the instrument

Example

<pre>print(localnode.model)</pre>
Outputs the model number of the local node. For example: DAQ6510

Also see

[localnode.serialno](#) (on page 14-360)

localnode.password

This attribute stores the instrument password.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	No	Rear-panel LAN reset	Nonvolatile memory	"admin"

Usage

```
localnode.password = "password"
```

<i>password</i>	A string that contains the instrument password (maximum 30 characters)

Details

When the access to the instrument is set to protected or lockout, this is the password that is used to gain access.

If you forget the password, you can reset the password to the default:

1. On the front panel, press **MENU**.
2. Under System, select **Info/Manage**.
3. Select **Password Reset**.

Example

<code>localnode.password = "N3wpa55w0rd"</code>	Changes the password to N3wpa55w0rd.

Also see

[localnode.access](#) (on page 14-355)

localnode.prompts

This attribute determines if the instrument generates prompts in response to command messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	localnode.DISABLE

Usage

```
prompting = localnode.prompts  
localnode.prompts = prompting
```

<i>prompting</i>	Do not generate prompts: <code>localnode.DISABLE</code> Generate prompts: <code>localnode.ENABLE</code>

Details

When the prompting mode is enabled, the instrument generates prompts when the instrument is ready to take another command. Because the prompt is not generated until the previous command completes, enabling prompts provides handshaking with the instrument to prevent buffer overruns.

When prompting is enabled, the instrument might generate the following prompts:

- **TSP>**. The standard prompt, which indicates that the previous command completed normally.
- **TSP?**. The prompt that is issued if there are unread entries in the event log when the prompt is issued. Like the TSP> prompt, it indicates that processing of the command is complete. It does not mean the previous command generated an error, only that there were still errors in the event log when command processing completed.
- **>>>>**. The continuation prompt, which occurs when downloading scripts. When downloading scripts, many command messages must be sent as a group. The continuation prompt indicates that the instrument is expecting more messages as part of the present command.

Commands do not generate prompts. The instrument generates prompts in response to command completion.

Prompts are enabled or disabled only for the remote interface that is active when you send the command. For example, if you enable prompts when the LAN connection is active, they are not enabled for a subsequent USB connection.

NOTE

Do not disable prompting when using Test Script Builder. Test Script Builder requires prompts and sets the prompting mode automatically. If you disable prompting, the instrument stops responding when you communicate using Test Script Builder because it is waiting for a common complete prompt from Test Script Builder.

Example

```
localnode.prompts = localnode.ENABLE
```

Enable prompting.

Also see

[tsplink.initialize\(\)](#) (on page 14-543)

localnode.prompts4882

This attribute enables and disables the generation of prompts for IEEE Std 488.2 common commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	localnode.ENABLE

Usage

```
prompting = localnode.prompts4882
localnode.prompts4882 = prompting
```

prompting

IEEE Std 488.2 prompting mode:

- Disable prompting: `localnode.DISABLE`
- Enable prompting: `localnode.ENABLE`

Details

When this attribute is enabled, the IEEE Std 488.2 common commands generate prompts if prompting is enabled with the `localnode.prompts` attribute. If `localnode.prompts4882` is enabled, limit the number of `*trg` commands sent to a running script to 50 regardless of the setting of the `localnode.prompts` attribute.

When this attribute is disabled, IEEE Std 488.2 common commands do not generate prompts. When using the `*trg` command with a script that executes `trigger.wait()` repeatedly, disable prompting to avoid problems associated with the command interface input queue filling.

Example

```
localnode.prompts4882 = localnode.DISABLE
```

Disables IEEE Std 488.2 common command prompting.

Also see

[localnode.prompts](#) (on page 14-358)

localnode.serialno

This attribute stores the serial number of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
serialno = localnode.serialno
```

serialno

The serial number of the instrument

Details

This indicates the instrument serial number.

Example

```
display.clear()
display.settext(display.TEXT2, "Serial #: " ..localnode.serialno)
display.changescreen(display.SCREEN_USER_SWIPE)
```

Clears the instrument display.
Places the serial number of this instrument on the bottom line of the USER swipe screen display. Displays the USER swipe screen.

Also see

[localnode.model](#) (on page 14-357)
[localnode.version](#) (on page 14-363)

localnode.settime()

This function sets the date and time of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.settime()
localnode.settime(year, month, day, hour, minute, second)
localnode.settime(hour, minute, second)
localnode.settime(os.time({year, month, day}))
localnode.settime(os.time({year = year, month = month, day = day, hour = hour, min
    = minute, sec = second}))
```

<i>year</i>	Year; must be more than 1970
<i>month</i>	Month (1 to 12)
<i>day</i>	Day (1 to 31)
<i>hour</i>	Hour in 24-hour time format (0 to 23)
<i>minute</i>	Minute (0 to 59)
<i>second</i>	Second (0 to 59)

Details

Internally, the instrument bases time in UTC time. UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

When called without a parameter (the first form), the function returns the current time.

Example 1

```
localnode.settime(2017, 12, 5, 15, 48, 20)
print(localnode.settime())
```

Sets the date and time to December 5, 2017 at 3:48:20 pm and verifies the time.

Output:

```
Tue Dec 5 15:48:20 2017
```

Example 2

```
systemTime = os.time({year = 2018,
    month = 3,
    day = 31,
    hour = 14,
    min = 25})
localnode.settime(systemTime)
print(os.date('%c', gettime()))
```

Sets the date and time to Mar 31, 2018 at 2:25 pm.

Output:

```
Sat Mar 31 14:25:00 2018
```

Also see

[localnode.gettime\(\)](#) (on page 14-356)

localnode.showevents

This attribute sets whether or not the instrument automatically outputs generated events to the remote interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle	Not saved	0 (no events sent)

Usage

```
errorMode = localnode.showevents
localnode.showevents = errorMode
```

errorMode

The errors that are returned:

- No events: 0
- Errors only: 1 (eventlog.SEV_ERROR)
- Warnings only: 2 (eventlog.SEV_WARN)
- Errors and warnings: 3 (eventlog.SEV_ERROR | eventlog.SEV_WARN)
- Information only: 4 (eventlog.SEV_INFO)
- Information and errors: 5 (eventlog.SEV_INFO | eventlog.SEV_ERROR)
- Warnings and information: 6 (eventlog.SEV_INFO | eventlog.SEV_WARN)
- All events: 7 (eventlog.SEV_ALL)

Details

Enable this attribute to have the instrument output generated events to the remote interface.

Events are output after a command message is executed but before prompts are issued (if prompts are enabled with `localnode.prompts`).

If this attribute is disabled, use `eventlog.next()` to retrieve unread events from the event log.

Events are enabled or disabled only for the remote interface that is active when you send the command. For example, if you enable show events when the GPIB connection is active, they will not be enabled for a subsequent USB connection.

Example

```
localnode.showevents = eventlog.SEV_ERROR | eventlog.SEV_INFO
trigger.digin[3].edge = trigger.EDGE_EITHER
```

Send generated error and warning messages.

Example output if the edge cannot be sent to either:

```
1805, Settings conflict: setting input edge when line 3 set for digital
```

Also see

[eventlog.clear\(\)](#) (on page 14-328)

[localnode.prompts](#) (on page 14-358)

localnode.version

This attribute stores the firmware version of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
version = localnode.version
```

<code>version</code>	Instrument version level

Details

This attribute indicates the version number of the firmware that is presently running in the instrument.

Example

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

Outputs the present version level. Example output:
1.0.0a

Also see

[localnode.model](#) (on page 14-357)

[localnode.serialno](#) (on page 14-360)

node[N].execute()

This function starts test scripts on a remote TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

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

<i>N</i>	The node number of this instrument: 1 to 63
<i>scriptCode</i>	A string containing the source code

Details

This command is only applicable to TSP-Link systems. You can use this command to use the remote master node to run a script on the specified node. This function does not run test scripts on the master node; only on the subordinate node when initiated by the master node.

This function may only be called when the group number of the node is different than the node of the master.

This function does not wait for the script to finish execution.

Example 1

<pre>node[2].execute(sourcecode)</pre>
Runs script code on node 2. The code is in a string variable called <code>sourcecode</code> .

Example 2

<pre>node[3].execute("x = 5")</pre>
Runs script code in string constant (<code>x = 5</code>) to set <code>x</code> equal to 5 on node 3.

Example 3

<pre>node[32].execute(TestDut.source)</pre>
Runs the test script stored in the variable <code>TestDut</code> (previously stored on the master node) on node 32.

Also see

[tsplink.group](#) (on page 14-543)

node[N].getglobal()

This function returns the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = node[N].getglobal("name")
```

value	The value of the variable
N	The node number of this instrument: 1 to 63
name	The global variable name

Details

This function retrieves the value of a global variable from the runtime environment of this node.

Do not use this command to retrieve the value of a global variable from the local node. Instead, access the global variable directly. This command should only be used from a remote master when controlling this instrument over a TSP-Link network.

Example

```
print(node[5].getglobal("test_val"))
```

Retrieves and outputs the value of the global variable named `test_val` from node 5.

Also see

[node\[N\].setglobal\(\)](#) (on page 14-366)

node[N].setglobal()

This function sets the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
node[N].setglobal("name", value)
```

<i>N</i>	The node number of this instrument: 1 to 63
<i>name</i>	The global variable name to set
<i>value</i>	The value to assign to the variable

Details

From a remote node, use this function to assign the given value to a global variable.

Do not use this command to create or set the value of a global variable from the local node (set the global variable directly instead). This command should only be used from a remote master when controlling this instrument over a TSP-Link network.

Example

<pre>node[3].setglobal("x", 5)</pre>
Sets the global variable <code>x</code> on node 3 to the value of 5.

Also see

[node\[N\].getglobal\(\)](#) (on page 14-365)

opc()

This function sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

opc ()

Details

This function causes the operation complete bit in the Status Event Status Register to be set when all previously started local overlapped commands are complete.

Note that each node independently sets its operation complete bits in its own status model. Any nodes that are not actively performing overlapped commands set their bits immediately. All remaining nodes set their own bits as they complete their own overlapped commands.

Example

<pre>opc () waitcomplete () print ("1")</pre>	Output: 1

Also see

- [*OPC](#) (on page 15-9)
- [Status model](#) (on page 16-1)
- [waitcomplete\(\)](#) (on page 14-572)

print()

This function generates a response message.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
print(value1)
print(value1, value2)
print(value1, ..., valueN)
```

value1	The first argument to output
value2	The second argument to output
valueN	The last argument to output
...	One or more values separated with commas

Details

TSP-enabled instruments do not have inherent query commands. Like other scripting environments, the `print()` command and other related `print()` commands generate output. The `print()` command creates one response message.

The output from multiple arguments is separated with a tab character.

Numbers are printed using the `format.asciiprecision` attribute. If you want use Lua formatting, print the return value from the `tostring()` function.

Example 1

```
x = 10
print(x)
```

Example of an output response message:
10

Note that your output might be different if you set your ASCII precision setting to a different value.

Example 2

```
x = true
print(tostring(x))
```

Example of an output response message:
true

Also see

[format.asciiprecision](#) (on page 14-341)

printbuffer()

This function prints data from tables or reading buffer subtables.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printbuffer(startIndex, endIndex, bufferVar)
printbuffer(startIndex, endIndex, bufferVar, bufferVar2)
printbuffer(startIndex, endIndex, bufferVar, ..., bufferVarN)
```

<i>startIndex</i>	Beginning index of the buffer to print; this must be more than one and less than <i>endIndex</i>
<i>endIndex</i>	Ending index of the buffer to print; this must be more than <i>startIndex</i> and less than the index of the last entry in the tables
<i>bufferVar</i>	Name of first table or reading buffer subtable to print; may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>bufferVar2</i>	Second table or reading buffer subtable to print; may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
<i>bufferVarN</i>	The last table or reading buffer subtable to print; may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer
...	One or more tables or reading buffer subtables separated with commas

Details

If *startIndex* is set to less than 1 or if *endIndex* is more than the size of the index, 9.910000e+37 is returned for each value outside the allowed index and an event is generated.

If overlapped commands use the specified reading buffers and the commands are not complete (at least to the specified index), this function outputs data as it becomes available.

When there are outstanding overlapped commands to acquire data, *n* refers to the index that the last entry in the table has after all the readings have completed.

If you pass a reading buffer instead of a reading buffer subtable, the default subtable for that reading buffer is used.

This command generates a single response message that contains all data.

The output of `printbuffer()` is affected by the data format selected by `format.data`. If you set `format.data` to `format.REAL32` or `format.REAL64`, you have fewer options for buffer elements. With these formats, the only buffer elements available are readings, relativetimestamps, and extravalues. If you request a buffer element that is not permitted for the selected data format, the instrument returns 9.91e37.

You can use the *bufferVar* attributes that are listed in the following table with the print buffer command. For example, if `testData` is the buffer, you can use `testData.dates` attribute to print the date of each reading in the `testData` buffer.

You can use *bufferVar.n* to retrieve the number of readings in the specified reading buffer.

Attribute	Description
<i>bufferVar.channels</i>	The channels that produced the readings stored in the reading buffer; see bufferVar.channels (on page 14-42)
<i>bufferVar.dates</i>	The dates of readings stored in the reading buffer; see bufferVar.dates (on page 14-44)
<i>bufferVar.extraformattedvalues</i>	The measurement and the unit of measure of the additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option; see bufferVar.extraformattedvalues (on page 14-46)
<i>bufferVar.extravalueunits</i>	The units of the additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option; see bufferVar.extravalueunits (on page 14-49)
<i>bufferVar.extravalues</i>	The additional values (such as the sense voltage from a DC voltage ratio measurement); the reading buffer style must be set to full to use this option; see bufferVar.extravalues (on page 14-48)
<i>bufferVar.formattedreadings</i>	The stored readings formatted as they appear on the front-panel display; see bufferVar.formattedreadings (on page 14-52)
<i>bufferVar.fractionalseconds</i>	The fractional portion of the timestamp (in seconds) of when each reading occurred; see bufferVar.fractionalseconds (on page 14-53)
<i>bufferVar.readings</i>	The readings stored in a specified reading buffer; see bufferVar.readings (on page 14-57)
<i>bufferVar.relativetimestamps</i>	The timestamps, in seconds, when each reading occurred relative to the timestamp of reading buffer entry number 1; see bufferVar.relativetimestamps (on page 14-58)
<i>bufferVar.seconds</i>	The nonfractional seconds portion of the timestamp when the reading was stored in UTC format; see bufferVar.seconds (on page 14-59)
<i>bufferVar.statuses</i>	The status values of readings in the reading buffer; see bufferVar.statuses (on page 14-61)
<i>bufferVar.times</i>	The time when the instrument made the readings; see bufferVar.times (on page 14-62)
<i>bufferVar.timestamps</i>	The timestamps of readings stored in the reading buffer; see bufferVar.timestamps (on page 14-63)
<i>bufferVar.units</i>	The unit of measure that is stored with readings in the reading buffer; see bufferVar.units (on page 14-65)

Example 1

```
reset()
dmm.measure.func = dmm.FUNC_DC_CURRENT
testData = buffer.make(200)
format.data = format.ASCII
format.asciiprecision = 6
trigger.model.load("SimpleLoop", 6, 0, testData)
trigger.model.initiate()
waitcomplete()
printbuffer(1, testData.n, testData.readings, testData.units,
            testData.relativetimestamps)
```

Reset the instrument.

Set the measure function to DC current.

Set the data format and ASCII precision.

Use trigger model SimpleLoop to create a 6-count loop with no delays that stores data in the reading buffer testBuffer.

Start the trigger model, wait for the commands to complete, and output the readings.

Use of `testData.n` (*bufferVar.n*) indicates that the instrument should output all readings in the reading buffer. In this example, `testBuffer.n` equals 6.

Example of output data:

```
1.10458e-11, Amp DC, 0.00000e+00, 1.19908e-11, Amp DC, 1.01858e-01, 1.19908e-11, Amp DC,
2.03718e-01, 1.20325e-11, Amp DC, 3.05581e-01, 1.20603e-11, Amp DC, 4.07440e-01, 1.20325e-
11, Amp DC, 5.09299e-01
```

Example 2

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

Using the same buffer created in Example 1, output the readings, units, and relative timestamps on a separate line for each reading.

```
1.10458e-11, Amp DC, 0.00000e+00
1.19908e-11, Amp DC, 1.01858e-01
1.19908e-11, Amp DC, 2.03718e-01
1.20325e-11, Amp DC, 3.05581e-01
1.20603e-11, Amp DC, 4.07440e-01
1.20325e-11, Amp DC, 5.09299e-01
```

Also see

[bufferVar.n](#) (on page 14-55)

[bufferVar.readings](#) (on page 14-57)

[format.asciiprecision](#) (on page 14-341)

[format.byteorder](#) (on page 14-342)

[format.data](#) (on page 14-343)

[printnumber\(\)](#) (on page 14-372)

printnumber()

This function prints numbers using the configured format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printnumber(value1)
printnumber(value1, value2)
printnumber(value1, ..., valueN)
```

value1	First value to print in the configured format
value2	Second value to print in the configured format
valueN	Last value to print in the configured format
...	One or more values separated with commas

Details

There are multiple ways to use this function, depending on how many numbers are to be printed.

This function prints the given numbers using the data format specified by `format.data` and `format.asciiprecision`.

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x, 2.54321, 3.1)
```

Configure the ASCII precision to 10 and set `x` to 2.54.

Read the value of `x` based on these settings.

Change the ASCII precision to 3.

View how the change affects the output of `x` and some numbers.

Output:

```
2.540000000e+00
2.54e+00, 2.54e+00, 3.10e+00
```

Also see

- [format.asciiprecision](#) (on page 14-341)
- [format.byteorder](#) (on page 14-342)
- [format.data](#) (on page 14-343)
- [print\(\)](#) (on page 14-368)
- [printbuffer\(\)](#) (on page 14-369)

reset()

This function resets commands to their default settings and clears the buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
reset()  
reset(system)
```

<i>system</i>	If the node is the master, the entire system is reset: <code>true</code> Only the local group is reset: <code>false</code>

Details

The `reset()` command in its simplest form resets the entire TSP-enabled system, including the controlling node and all subordinate nodes.

If you want to reset a specific instrument, use the `node[N].reset()` command. Also use the `node[N].reset()` command to reset an instrument on a subordinate node.

When no value is specified for *system*, the default value is `true`.

You can only reset the entire system using `reset(true)` if the node is the master. If the node is not the master node, executing this command generates an error event.

Example

<code>reset(true)</code>	If the node is the master node, the entire system is reset; if the node is not the master node, an error event is generated.

Also see

[Resets](#) (on page 3-83)

scan.add()

This function adds channels to the scan list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
scan.add("channelList")
scan.add("channelList", "configList")
scan.add("channelList", "configList", index)
```

<i>channelList</i>	List of channels to add, in the order in which they should occur in the scan
<i>configList</i>	A string that defines the configuration list to recall
<i>index</i>	The index in the configuration list to recall; default is 1

Details

Use this function to add channels to the present scan list. If the scan list does not exist, it also creates a scan list.

Channels are added to the end of the present list in the order in which they are specified in the channel list.

If you include a configuration list, the configuration list must exist before you send this command.

NOTE

The front-panel SCAN screen does not show settings set by this configuration list parameter. To check settings, use the command `scan.list()`.

Example 1

```
scan.create("101:104")
scan.add("111, 110, 109")
scan.add("107")
```

Replaces the existing scan list with a scan list that scans channels 1 to 4 on slot 1. Adds channels 11, 10, and 9 to the end of the scan list, to be scanned in that order.

Adds channel 7 to the end of the scan list.

Example 2

```
reset()
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
dmm.measure.range = 10
dmm.measure.nplc = 10

dmm.measure.configlist.create("scanconfig")
dmm.measure.configlist.store("scanconfig")
dmm.measure.nplc = 0.01
dmm.measure.configlist.store("scanconfig")

channel.setdmm("101:104", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:104", dmm.ATTR_MEAS_NPLC, 0.1)
scan.create("101:104")
scan.add("111, 110, 109", "scanconfig")
scan.add("107", "scanconfig", 2)
```

Set up the DMM for the settings you want to use in the scan. This example shows the function set to DC voltage, with a measurement range of 10 V and the NPLCs set to 10.

Create a configuration list named `scanconfig`.

Store the present configuration to `scanconfig`.

Set NPLC to 0.01.

Store the present configuration to `scanconfig`. This configuration is stored in index 2.

Set up channels 1 to 4 on slot 1 for DC voltage with NPLC set to 0.1.

Create a scan list with a scan list that includes channels 1 to 4.

Adds channels 11, 10, and 9 to the end of the scan list, to be scanned in that order. The settings in the configuration list `scanconfig` are used for these channels.

Adds channel 7 to the end of the scan list. The settings in index 2 of `scanconfig` are used for this channel.

Also see

[dmm.measure.configlist.create\(\)](#) (on page 14-214)

[scan.create\(\)](#) (on page 14-383)

[scan.list\(\)](#) (on page 14-389)

[Scanning and triggering](#) (on page 5-29)

scan.addsinglestep()

This function allows you to include multiple channels in a single scan step.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
scan.addsinglestep("channelList")
scan.addsinglestep("channelList", configList)
scan.addsinglestep("channelList", configList, index)
```

<i>channelList</i>	List of channels to add, in the order in which they should occur in the scan
<i>configList</i>	A string that defines the configuration list to recall
<i>index</i>	The index in the configuration list to recall; default is 1

Details

This command adds a list of channels to be closed simultaneously in a single step of a scan.

If you need to make measurements using multiple functions on these channels, you can use the configuration list parameter to call the function settings. The configuration list must be created before calling it in this command.

NOTE

The front-panel SCAN menu does not show settings set by the configuration list parameter. To check settings, use the command `scan.list()`.

Example

```
scan.create("101:104")
scan.addsinglestep("109, 106, 103")
scan.add("107")
print(scan.list())
```

Replaces the existing scan list with a scan list that scans channels 1 to 4 on slot 1.

Adds channels 9, 6, and 3 to the end of the scan list, to be scanned in that order.

Adds channel 7 to the end of the scan list.

The scan list returns:

```
INIT:   OPEN:
      101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,12
      0,121,122,125,124,123      CLOSE:
1:      OPEN:   CLOSE: 101      None
2:      OPEN: 101      CLOSE: 102      None
3:      OPEN: 102      CLOSE: 103      None
4:      OPEN: 103      CLOSE: 104      None
5:      OPEN: 104      CLOSE: 109,106,103      None
6:      OPEN: 109,106,103      CLOSE: 107      None
```

Also see

[scan.create\(\)](#) (on page 14-383)

[scan.list\(\)](#) (on page 14-389)

[Scanning and triggering](#) (on page 5-29)

scan.addwrite()

This function writes a specified value to a channel at the added step in the scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.addwrite("channelList", writeValue)
```

<i>channelList</i>	String specifying channels to add using normal channel list syntax
<i>writeValue</i>	The value to write to the channel for this scan step

Details

This command is similar to issuing `channel.write()` at the scan step. Specifying multiple channels in *channelList* causes multiple steps to be added to the scan. Allows you to write the DAC or digital I/O in a scan.

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.

Example

<pre>scan.addwrite("124", 2)</pre>	Assume a 7706 in slot 1. Add a write step to DAC channel 24 to output 2 V.

Also see

[Scanning and triggering](#) (on page 5-29)

scan.alarmnotify

This attribute determines if the scan sends a trigger event when a value is out of limits.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	scan.OFF

Usage

```
state = scan.alarmnotify
scan.alarmnotify = state
```

<i>value</i>	Determine whether to generate an alarm notify trigger: - scan.OFF: Do not send notify trigger events. - scan.ON: Send a notify trigger event when a value is out of limits.

Details

When this is set on, a trigger is generated when the measurements exceed the limits set for the channels in the scan. To use this trigger, set a stimulus to `trigger.EVENT_SCAN_ALARM_LIMIT`.

Example

<pre>reset() channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE) scan.add("101:129") scan.learnlimits(0.25, 3) trigger.extout.stimulus = trigger.EVENT_SCAN_ALARM_LIMIT scan.alarmnotify = scan.ON</pre>
Reset the instrument. Set channels 1 to 9 to the DCV measure function. Create a scan that includes channels 1 to 9. Use learn limits to establish the limits to within 25% over three iterations of scan limits. Enable alarm notification on the external trigger out line.

Also see

None

scan.buffer

This attribute defines which buffer is used with the scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	defbuffer1

Usage

```
scan.buffer = readingBuffer
readingBuffer = scan.buffer
```

readingBuffer	The reading buffer to use to collect data from the scan

Details

This selects the buffer that stores the data generated by the scan.

Example

```
reset()
channel.setdmm("101:107", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
scan.create("101:107")
scan.buffer = defbuffer2
trigger.model.initiate()
waitcomplete()

if scan.buffer == defbuffer1 then
    print("defbuffer1")
else if scan.buffer == defbuffer2 then
    print("defbuffer2")
end
```

Sets the buffer for the scan to defbuffer2.
Verifies buffer name.

Also see

[Scanning and triggering](#) (on page 5-29)

scan.bypass

This attribute indicates whether the first channel of the scan waits for the channel stimulus event to be satisfied before closing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	scan.OFF

Usage

```
bypass = scan.bypass
scan.bypass = bypass
```

bypass	Disable or enable the bypass: ■ Disable: scan.OFF ■ Enabled: scan.ON

Details

When bypass is set to on and the channel stimulus for the scan is set to wait for a stimulus, the first channel of the scan closes when the scan starts (the stimulus setting is ignored).

For other channels, the channel stimulus must be satisfied before the channel action takes place.

When bypass is set to off, every channel (including the first) must satisfy the channel stimulus setting before the channel action occurs for that step.

Example

```
scan.bypass = scan.ON
print(scan.bypass)
```

Enables the bypass option for scanning and displays the present bypass state.

Output:

```
scan.ON
```

Also see

- [scan.channel.stimulus](#) (on page 14-381)
- [Scanning and triggering](#) (on page 5-29)

scan.channel.stimulus

This attribute determines which trigger event causes the channel action to occur.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	trigger.EVENT_NONE

Usage

```
eventID = scan.channel.stimulus
scan.channel.stimulus = eventID
```

<i>eventID</i>	Trigger stimulus used for the channel action; see Details for trigger event IDs

Details

Set the event ID to one of the options in the following table.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	trigger.EVENT_NONE
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	trigger.EVENT_COMMAND
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	trigger.EVENT_DIGION
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	trigger.EVENT_TSPLINKN
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	trigger.EVENT_LANN

Trigger events	
Analog trigger	trigger.EVENT_ANALOGTRIGGER
Trigger event blender N (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
Trigger timer N (1 to 4) expired	trigger.EVENT_TIMERN
External in trigger	trigger.EVENT_EXTERNAL
Scan alarm limit exceeded	trigger.EVENT_SCAN_ALARM_LIMIT
Channel closed	trigger.EVENT_SCAN_CHANNEL_READY (returns trigger.EVENT_NOTIFY6)
Scan completed	trigger.EVENT_SCAN_COMPLETE (returns trigger.EVENT_NOTIFY8)
Measure completed	trigger.EVENT_SCAN_MEASURE_COMPLETE (returns trigger.EVENT_NOTIFY7)
Limit value for scan reached	trigger.EVENT_SCAN_ALARM_LIMIT (returns trigger.EVENT_NOTIFY3)

Example

```

reset()
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:105", dmm.ATTR_MEAS_NPLC, 0.01)
scan.channel.stimulus = trigger.EVENT_TIMER1
scan.create("101:105")

trigger.timer[1].count = 0
trigger.timer[1].delay = 1
trigger.timer[1].enable = trigger.ON

timer.cleartime()
trigger.model.initiate()
waitcomplete()

```

Reset the instrument.
Set up channels 1 to 5 in slot 1 to measure DC voltage with an NPLC of 0.01.
Set the channel stimulus to event timer 1.
Create a scan list that includes channels 1 to 5 in slot 1.
Set the trigger timer to generate trigger events indefinitely.
Set the delay between triggers to 1 s.
Turn the trigger timer on.
Reset the timer to 0 s.
Start the trigger model.

Also see

[Scanning and triggering](#) (on page 5-29)

scan.create()

This function deletes the existing scan list and creates a new list of channels to scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
scan.create()  
scan.create("channelList")  
scan.create("channelList", configList)  
scan.create("channelList", configList, index)
```

<i>channelList</i>	String specifying channels to add to the new scan list
<i>configList</i>	A string that defines the configuration list to recall
<i>index</i>	The index in the configuration list to recall; default is 1

Details

The items in the channel list are scanned in the order listed.

Sending this command with no parameters clears the existing scan list.

Using a configuration list allows you to set multiple functions for the channels using the settings in the configuration list. The configuration list must exist before you send this command.

NOTE

The front-panel SCAN menu does not show settings set by the configuration list parameter. To check settings, use the command `scan.list()`.

Example 1

```
scan.create("101:109")
```

Replaces the active scan list with an empty scan list.
Adds channels 1 through 9 on slot 1. Uses the existing DMM configuration.

Example 2

```
scan.create()  
  
for chan = 101, 109 do  
    scan.add(tostring(chan))  
end
```

Replaces the active scan list with an empty scan list.
Loops through channels 1 to 9, and then adds 9 channels to the scan list. The parameter `(tostring(chan))` converts the channel number to a string.
The scan list now has, in order, channels 1 through 9 on slot 1.
Uses the existing DMM configuration.

Example 3

```
reset()
defbuffer1.clear()
defbuffer1.capacity = 100
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE, 10)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 0.1)
channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_5_5)

scan.create("101:109")
scan.sancount = 10
scan.scaninterval = 1.0

trigger.model.initiate()
waitcomplete()

dmm.measure.read(defbuffer1)
printbuffer(1, defbuffer1.n, defbuffer1)
```

Reset the instrument.

Clear defbuffer1 and set it to 100 readings.

Set channels 1 to 9 on slot 1 to make DC voltage measurements on the 10 V range at 0.1 PLC.

Display 5.5 digits.

Create a scan of channels 1 to 9.

Set the scan count to 10.

Provide a one second delay between each scan.

Start the scan.

Read and output the data from the scan.

Example 4

```
reset()
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 1)
dmm.measure.configlist.create("dmm_active")
dmm.measure.configlist.store("dmm_active")

dmm.measure.func = dmm.FUNC_RESISTANCE
dmm.measure.nplc = 0.5
dmm.measure.configlist.store("dmm_active")

scan.create("101:109")
scan.add("101:109", "dmm_active", 2)
trigger.model.initiate()
waitcomplete()
printbuffer(1, defbuffer1.n, defbuffer1, defbuffer1.units)
```

This example demonstrates how to use a configuration list to change the scan settings.
Set nine channels to DC voltage with an NPLC of 1.
Create a configuration list named `dmm_active`.
Save the settings to `dmm_active`.
Set the function to 2-wire resistance and the NPLC to 0.5.
Save the settings to `dmm_active`.
Create a scan of channels 1 to 9.
Add channels 1 to 9 to the scan, using the settings in `dmm_active`, index 2.
Start the scan and output the data.

Example 5

```
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 1)

channel.setdmm("111:120", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_RESISTANCE)
channel.setdmm("111:120", dmm.ATTR_MEAS_NPLC, 1)

scan.create("101,111,120")
trigger.model.initiate()
```

This example sets DMM settings on twenty channels and then creates a scan for three of the channels.

Also see

[scan.add\(\)](#) (on page 14-374)
[scan.list\(\)](#) (on page 14-389)
[Scanning and triggering](#) (on page 5-29)
[Configuration lists](#) (on page 4-99)

scan.export()

This command stores data from a scan to a file on a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.export("/usb1/filename", when)
scan.export("/usb1/filename", when, what)
```

<i>filename</i>	The name of the file to be created on the USB flash drive
<i>when</i>	When to write the data to the file: ■ scan.WRITE_AFTER_STEP: At completion of each scan step ■ scan.WRITE_AFTER_SCAN: At completion of each scan ■ scan.WRITE_AT_END: At completion of all scans ■ scan.WRITE_NEVER: Do not write data to a file
<i>what</i>	Which data to include; see Details for options

Details

This command sets up the instrument to export scan data. If an option to export data is selected, data is sent to a USB flash drive inserted into the USB port on the front panel of the instrument. Export files are limited to 500 MB. When data exceeds 500 MB, another file is created with *_n* added to the file name, where *n* starts at 1 and is incremented for each additional file.

The file name must specify the full path (including /usb1/). If included, the file extension must be set to .csv. If no file extension is specified, .csv is added.

The exported data is time-stamped.

Exporting data can impact scan performance. The more often exports occur, the more the impact on performance. Therefore, exporting data at completion of each step results in the slowest performance.

The DAQ6510 does not check for existing files when you save. Verify that you are using a unique name to avoid overwriting any existing .csv files on the flash drive.

You can OR the `buffer.COL_CHANNEL`, `buffer.COL_CSV_CHAN_COLS`, and `buffer.COL_CSV_EASY_GRAPH` options with the timestamp options (`buffer.COL_TIME_ABSOLUTE`, `buffer.COL_TIME_PARTS`, `buffer.COL_TIME_RAW`, `buffer.COL_TIME_RELATIVE`, and `buffer.COL_TIMESTAMP_READING`).

You cannot use `buffer.COL_CSV_CHAN_COLS` if *when* is set to `scan.WRITE_AFTER_STEP`.

You cannot use `buffer.COL_CSV_EASY_GRAPH` if *when* is set to `scan.WRITE_AFTER_STEP` or `scan.WRITE_AFTER_SCAN`.

Option	Export includes
<code>buffer.COL_ALL</code>	All data
<code>buffer.COL_BRIEF</code>	Reading and relative time
<code>buffer.COL_CHANNEL</code>	Channel

<code>buffer.COL_CSV_CHAN_COLS</code>	Ignore other columns and use special format with a column per channel
<code>buffer.COL_CSV_EASY_GRAPH</code>	Ignore other columns and use special format that is easy to graph in Microsoft Excel
<code>buffer.COL_DISPLAY_DIGITS</code>	The setting for the display digits
<code>buffer.COL_EXTRA</code>	Relative time and additional values if they exist (such as the sense voltage from a DC voltage ratio measurement)
<code>buffer.COL_EXTRA_RANGE</code>	Extra value range digits
<code>buffer.COL_EXTRA_UNIT</code>	Extra value units
<code>buffer.COL_EXTRA_VALUE</code>	Extra value
<code>buffer.COL_INDEX</code>	Index into buffer
<code>buffer.COL_LIMITS</code>	The status of all limits
<code>buffer.COL_MATH</code>	Math enabled (F if math is not enabled; T if math is enabled) and relative time
<code>buffer.COL_ORIGIN</code>	Origin status
<code>buffer.COL_QUESTIONABLE</code>	Questionable status
<code>buffer.COL_RANGE_DIGITS</code>	Range digits
<code>buffer.COL_READING</code>	The measurement reading
<code>buffer.COL_STANDARD</code>	The relative time, reading, channel, and source value
<code>buffer.COL_START</code>	Status of start group
<code>buffer.COL_STATUS</code>	The status information associated with the measurement; see the "Buffer status bits for sense measurements" table in bufferVar.statuses (on page 14-61)
<code>buffer.COL_TERMINAL</code>	Terminal status
<code>buffer.COL_TIME_ABSOLUTE</code>	The time when the data point was measured as an absolute timestamp
<code>buffer.COL_TIME_PARTS</code>	Absolute time in multiple columns
<code>buffer.COL_TIME_RAW</code>	Absolute time in seconds
<code>buffer.COL_TIME_RELATIVE</code>	The relative time when the data point was measured in seconds
<code>buffer.COL_TIMESTAMP_READING</code>	The timestamp reading
<code>buffer.COL_UNIT</code>	The reading and the unit of measure

Example

```
scan.export("/usb1/scandata", scan.WRITE_AFTER_STEP)
trigger.model.initiate()
```

Set up the instrument to export data from a scan to a file named `scandata` at the completion of each scan step.
Start the scan.

Also see

None

scan.learnlimits()

This function calculates alarm limits based on the present configuration of the system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.learnlimits(window)
scan.learnlimits(window, iterations)
```

<i>window</i>	Percentage of deviation from the measurement that is within limits: 0.1 to 1000
<i>iterations</i>	Number of times to run the scan to set limits: 1 to 10; default 1

Details

Auto Learn runs a scan and establishes alarm limits based on the measurements from the scan. Make sure your system is in a stable state before running Auto Learn.

Example

```
scan.create("101:105")
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
scan.learnlimits(1, 5)
print(channel.getdmm("101:105", dmm.ATTR_MEAS_LIMIT_HIGH_1))
print(channel.getdmm("101:105", dmm.ATTR_MEAS_LIMIT_LOW_1))
```

Create a scan on channels 101 to 105. Set up the channels to measure DC voltage. Start the learn limits calculation, with a 1% window and 5 iterations. Output the values for limit 1 high and low.

Also see

None

scan.list()

This function returns a list that includes the initial open or close state of any cards installed in the instrument and the settings at each step of the scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No	Instrument reset Power cycle	Not applicable	Not applicable

Usage

```
scanList = scan.list()
```

<i>scanList</i>	A string that lists information for each scan step

Details

This command lists the existing scan list, including each step in the scan and information for step, open, or close status.

If the scan list is empty, the return is `EMPTY`.

Example

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:105", dmm.ATTR_MEAS_NPLC, 0.01)
channel.setdmm("201:202", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_AC_VOLTAGE)
scan.create("101:105, 201, 202")
print(scan.list())
```

With a 7701 in slot 1 and a 7708 in slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01. Set up channels 1 and 2 on slot 2 to measure AC voltage.

Create a scan that includes channels 101 to 105 and 201 to 202.

Print the scan list. The output looks similar to the following code.

```
INIT:   OPEN:
101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,12
0,121,122,123,124,125,126,127,128,129,130,131,132,135,134,133,201,202,203,204,
205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,22
4,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,243,242,241
CLOSE:
1: OPEN:      CLOSE: 101,135      Voltage DC
2: OPEN: 101 CLOSE: 102      Voltage DC
3: OPEN: 102 CLOSE: 103      Voltage DC
4: OPEN: 103 CLOSE: 104      Voltage DC
5: OPEN: 104 CLOSE: 105      Voltage DC
6: OPEN: 135,105 CLOSE: 201,243      Voltage AC
7: OPEN: 201 CLOSE: 202      Voltage AC
```

Also see

[scan.add\(\)](#) (on page 14-374)

[scan.addsinglestep\(\)](#) (on page 14-376)

[scan.create\(\)](#) (on page 14-383)

[Scanning and triggering](#) (on page 5-29)

scan.measure.interval

This attribute specifies the interval time between measurement requests.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	0

Usage

```
time = scan.measure.interval  
scan.measure.interval = time
```

time	The interval time between measurements (0 to 100 ks)

Details

This command specifies the time between measurements in the scan.

Example

```
scan.create("101:109")  
scan.scancount = 10  
scan.measure.interval = 1.0
```

Create a scan of channels 101 to 109.
Set the scan count to 10.
Provide a one second delay between each measurement in the scan.

Also see

- [Scanning and triggering](#) (on page 5-29)
- [scan.scancount](#) (on page 14-399)

scan.measure.stimulus

This attribute selects the trigger for the measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	trigger.EVENT_NONE

Usage

```
eventID = scan.measure.stimulus
scan.measure.stimulus = eventID
```

eventID	The event that triggers the measurement

Details

Use this to start a set of measurement count readings that are triggered by a single event.

The available trigger events are described in the following table.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	trigger.EVENT_NONE
Front-panel TRIGGER key press	trigger.EVENT_DISPLAY
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	trigger.EVENT_NOTIFYN
A command interface trigger (bus trigger): - Any remote interface: *TRG - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	trigger.EVENT_COMMAND
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	trigger.EVENT_DIGION
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	trigger.EVENT_TSPLINKN
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	trigger.EVENT_LANN

Trigger events	
Analog trigger	trigger.EVENT_ANALOGTRIGGER
Trigger event blender N (1 to 2), which combines trigger events	trigger.EVENT_BLENDERN
Trigger timer N (1 to 4) expired	trigger.EVENT_TIMERN
External in trigger	trigger.EVENT_EXTERNAL
Scan alarm limit exceeded	trigger.EVENT_SCAN_ALARM_LIMIT
Channel closed	trigger.EVENT_SCAN_CHANNEL_READY (returns trigger.EVENT_NOTIFY6)
Scan completed	trigger.EVENT_SCAN_COMPLETE (returns trigger.EVENT_NOTIFY8)
Measure completed	trigger.EVENT_SCAN_MEASURE_COMPLETE (returns trigger.EVENT_NOTIFY7)
Limit value for scan reached	trigger.EVENT_SCAN_ALARM_LIMIT (returns trigger.EVENT_NOTIFY3)

Example

```

reset()
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
scan.add("101:109")
scan.restart = scan.ON
scan.bypass = scan.ON
scan.scancount = 10
scan.start.stimulus = trigger.EVENT_EXTERNAL
scan.channel.stimulus = trigger.EVENT_LAN1
scan.measure.stimulus = trigger.EVENT_DISPLAY
scan.scaninterval = 0.5
scan.measure.interval = 1
scan.monitor.channel = "102"
scan.monitor.mode = scan.MODE_HIGH
scan.monitor.limit.high.value = 10

```

Reset the instrument.
 Set up channels 101 to 109 to measure on the DC voltage function.
 Add channels 101 to 109 to a scan.
 Set the scan restart on power loss option on.
 Set the scan bypass on first scan option on.
 Set the scan count to 10.
 Set the scan start stimulus to the rear-panel EXTERNAL TRIGGER IN terminal.
 Set the channel stimulus to LAN line 1.
 Set the measure stimulus to the front-panel TRIGGER key.
 Set the scan interval to 1 s.
 Set the monitor channel to 102.
 Set the scan mode to start the scan when the measurements reach a high value of 10 V.

Also see

[Scanning and triggering](#) (on page 5-29)

scan.mode

This attribute sets the relay action when the scan starts.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	scan.MODE_OPEN_ALL

Usage

```
scanModeSetting = scan.mode  
scan.mode = scanModeSetting
```

<i>scanModeSetting</i>	Relay action when the scan starts: - scan.MODE_OPEN_ALL - scan.MODE_OPEN_USED: See Details - scan.MODE_FIXED_ABR: Automatic backplane relay; see Details

Details

When this attribute is set to open all, all channels on all slots are opened before a scan starts.

When the mode is set to open used, an intelligent open is performed. For channels that are not set to a function:

- All channels used in scanning are opened
- Closed channels not used in scanning remain closed during the scan

If any step is set to a function:

- Any commonside ohms backplane relays are opened on all slots
- Any amp channels are opened on all slots
- All channels and backplane relays involved in scanning are opened
- If a closed channel or backplane relay is not involved in scanning, it remains closed during the scan
- All channels are opened on any bank that contains backplane relays that are involved in scanning

When this attribute is set to automatic backplane relay, it is equivalent to setting open used, except that all required backplane relays are closed before the start of the scan. These backplane relays are not opened or closed during the scan and do not open at the end of the scan.

Example

<pre>scan.mode = scan.MODE_OPEN_USED</pre>	Sets the scan mode setting to open channels that are used in the scan.

Also see

[Scanning and triggering](#) (on page 5-29)

scan.monitor.channel

This attribute defines which channel to monitor for a limit to be reached before starting the scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	101

Usage

```
value = scan.monitor.channel
scan.monitor.channel = value
```

value	The channel to monitor

Details

The channel to monitor for a limit to be reached before starting the scan.

Example

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:105", dmm.ATTR_MEAS_NPLC, 0.01)
channel.setdmm("201", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
scan.monitor.limit.high.value = 28
scan.create("101:105")
scan.monitor.mode = scan.MODE_HIGH
trigger.model.initiate()
```

With 7700 switching modules in slot 1 and slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01.
Set up channel 1 in slot 2 to measure temperature.
Set the monitoring channel to be 201.
Set the high limit to 28 °C.
Create a scan that includes data for channels 101 to 105.
Set the scan monitor function to allow scanning to start when the measured value exceeds the defined high limit value (28 °C).
Initiate the scan, which starts when the temperature of channel 201 is above 28 °C.

Also see

[scan.monitor.limit.high.value](#) (on page 14-395)
[scan.monitor.limit.low.value](#) (on page 14-396)
[scan.monitor.mode](#) (on page 14-397)

scan.monitor.limit.high.value

This attribute specifies the high limit to be used by the scan monitor.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	0

Usage

```
value = scan.monitor.limit.high.value
scan.monitor.limit.high.value = value
```

value	The value of the upper limit applied to the monitor channel

Details

This command sets the high limit for the monitor.

Example

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:105", dmm.ATTR_MEAS_NPLC, 0.01)
channel.setdmm("201", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
scan.monitor.limit.high.value = 28
scan.create("101:105")
scan.monitor.mode = scan.MODE_HIGH
trigger.model.initiate()
```

With 7700 switching modules in slot 1 and slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01.
Set up channel 1 in slot 2 to measure temperature.
Set the monitoring channel to be 201.
Set the high limit to 28 °C.
Create a scan that includes data for channels 101 to 105.
Set the scan monitor function to allow scanning to start when the measured value exceeds the defined high limit value (28 °C).
Initiate the scan, which starts when the temperature of channel 201 is above 28 °C.

Also see

- [scan.monitor.limit.low.value](#) (on page 14-396)
- [scan.monitor.mode](#) (on page 14-397)

scan.monitor.limit.low.value

This attribute defines the low limit to be used by the scan monitor.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	0

Usage

```
value = scan.monitor.limit.low.value
scan.monitor.limit.low.value = value
```

value	The value of the lower limit applied to the monitor channel

Details

This command sets the low limit for the monitor.

Example

```
channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)
channel.setdmm("101:105", dmm.ATTR_MEAS_NPLC, 0.01)
channel.setdmm("201", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE)
scan.monitor.limit.low.value = 18
scan.monitor.limit.high.value = 28
scan.create("101:105")
scan.monitor.mode = scan.MODE_WINDOW
trigger.model.initiate()
```

With 7700 switching modules in slot 1 and slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01.
Set up channel 1 in slot 2 to measure temperature.
Set the monitoring channel to be 201.
Set the monitor channel low limit to 18 °C.
Set the high limit to 28 °C.
Create a scan that includes data for channels 101 to 105.
Set the scan monitor function to allow scanning to start when the measured value exceeds the defined high limit value (28 °C).
Initiate the scan, which starts when the temperature of channel 201 is above 28 °C.

Also see

[scan.monitor.limit.high.value](#) (on page 14-395)
[scan.monitor.mode](#) (on page 14-397)

scan.monitor.mode

This attribute determines if a scan starts immediately when triggered or after measurements reach a set value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	scan.MODE_OFF

Usage

```
value = scan.monitor.mode
scan.monitor.mode = value
```

value	The value to use to start the scan: - scan.MODE_OFF: Start the scan without waiting for a specific value - scan.MODE_HIGH: Start the scan when the measurement exceeds the value set by scan.monitor.limit.high.value - scan.MODE_LOW: Start the scan when the measurement is below the value set by scan.monitor.limit.low.value - scan.MODE_OUTSIDE: Start the scan when the measurement is less than the lower limit or more than the upper limit. - scan.MODE_WINDOW: Start the scan when the measurement is between the lower limit and the upper limit

Details

This command determines if measurements are monitored to start a scan. If measurements are monitored, it also determines if the measurement triggers the start of the scan when it reaches a high value, low value, inside the high and low values, or outside the high and low values. The scan start values are stored in defbuffer2.

Example

<pre>channel.setdmm("101:105", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE) channel.setdmm("101:105", dmm.ATTR_MEAS_NPLC, 0.01) channel.setdmm("201", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_TEMPERATURE) scan.monitor.limit.high.value = 28 scan.create("101:105") scan.monitor.mode = scan.MODE_HIGH trigger.model.initiate()</pre>
With 7700 switching modules in slot 1 and slot 2, set up channels 1 to 5 on slot 1 to measure DC voltage with an NPLC of 0.01. Set up channel 1 in slot 2 to measure temperature. Set the monitoring channel to be 201. Set the high limit to 28 °C. Create a scan that includes data for channels 101 to 105. Set the scan monitor function to allow scanning to start when the measured value exceeds the defined high limit value (28 °C). Initiate the scan, which starts when the temperature of channel 201 is above 28 °C.

Also see

[scan.monitor.limit.high.value](#) (on page 14-395)
[scan.monitor.limit.low.value](#) (on page 14-396)
[scan.monitor.channel](#) (on page 14-394)

scan.restart

This function causes a scan to automatically restart if it was interrupted by a power failure.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	scan.OFF

Usage

```
state = scan.restart  
scan.restart = state
```

state

Select:

- scan.ON: Restart scan
- scan.OFF: Do not restart scan

Details

If the restart option is enabled, the scan settings are saved in memory immediately after the scan is triggered and before the scan operation begins. All scan settings, including watched channels, need to be set before the scan starts. Any changes that are made after the scan starts are not recalled if power is lost and the scan needs to restart.

If the restart option is enabled and power is lost, the scan restarts when power is restored. The scan setup that was in place when the scan started becomes the power-up setup. It takes precedence over any other power-up setup. If the scan completes successfully, the scan setup is removed as the power-up setup.

If the DAQ6510 detects that a card was changed during the power-up sequence, restart is set to off, the interrupted scan is not resumed, and an event is generated. The instrument starts up normally.

When a scan is automatically restarted, it is logged in the event log.

NOTE

If a restart occurs, the original reading buffer data is retained (per the settings of the reading buffer) and a new reading buffer is created.

Example

```
scan.restart = scan.ON
```

Set scan to restart when power to the instrument is restored.

Also see

None

scan.scancount

This attribute sets the number of times the scan is repeated.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	1

Usage

```
scanCount = scan.scancount
scan.scancount = scanCount
```

<i>scanCount</i>	The scan count value (1 to 100,000,000); set to <code>scan.COUNT_INFINITE</code> or 0 to set the scan to repeat until aborted

Details

The scan count attribute setting indicates how many times the scan list is iterated through before the scan completes.

Example

```
scan.create("101:109")
scan.scancount = 100
scan.scaninterval = 1.0 -- delay between scans
```

Create a scan that includes channels 1 to 9 of slot 1.
Set the scan count to 100.
Set the delay between scans to 1 s.

Also see

- [Scanning and triggering](#) (on page 5-29)
- [Trigger model](#) (on page 8-35)

scan.scaninterval

This attribute specifies the interval time between scan starts when the scan count is more than one.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	0 (0 s)

Usage

```
scanInterval = scan.scaninterval
scan.scaninterval = scanInterval
```

<i>scanInterval</i>	The scan interval (0 to 100 ks)

Details

If the scan interval is less than the time the scan takes to run, the next scan starts immediately when the first scan finishes.

Example

<code>scan.scaninterval = 5</code>	Sets the scan count to 5 s.

Also see

[Scanning and triggering](#) (on page 5-29)

scan.state()

This function provides the present state of a running background scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scanState, scanCount, stepCount = scan.state()
```

<i>scanState</i>	The present state of the scan running in the background. Possible states include: ■ <code>scan.EMPTY</code>: Scan is not set up ■ <code>scan.BUILDING</code>: The DAQ6510 is building the scan ■ <code>scan.RUNNING</code>: The scan is running the trigger model portion of the scan ■ <code>scan.STEPPING</code>: The scan is running the channel action portion of the scan ■ <code>scan.ABORTED</code>: The scan was canceled ■ <code>scan.PAUSED</code>: The scan was paused ■ <code>scan.FAILED</code>: The scan failed ■ <code>scan.SUCCESS</code>: The scan completed successfully
<i>scanCount</i>	The number of scans that have completed
<i>stepCount</i>	The number of steps that have completed

Details

Returns the state of the present scan, the scan count, and the step count.

The scan count is the number of the present iteration through the scan portion of the trigger model. This number does not increment until the scan begins. Therefore, if the instrument is waiting for an input to trigger a scan start, the scan count represents the previous number of scan iterations. If no scan has begun, the scan count is zero.

The step count is 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 instrument is waiting for an input to trigger a channel action, the step count represents the previous step. If no step has yet completed, the step count is 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.

The information from the scan state command may be delayed up to 100 ms from the actual state of the scan because of system resources used by the scan.

Example

```
trigger.model.initiate()
scanState, scanCount, stepCount = scan.state()
print(scanState, scanCount, stepCount)
```

Runs a scan in the background.
Check the present scan state.
View returned values.
Output shows that scan is running:
scan.RUNNING, 1, 2

Also see

[scan.mode](#) (on page 14-393)
[Scanning and triggering](#) (on page 5-29)

scan.start.stimulus

This attribute determines which event starts the scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Create configuration script	trigger.EVENT_NONE

Usage

```
eventID = scan.start.stimulus
scan.start.stimulus = eventID
```

eventID	Trigger stimulus used to start the scan; see Details

Details

The events that you can use to start the scan are described in the following table.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	<code>trigger.EVENT_NONE</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
A command interface trigger (bus trigger): - Any remote interface: <code>*TRG</code> - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Scan alarm limit exceeded	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Channel closed	<code>trigger.EVENT_SCAN_CHANNEL_READY</code> (returns <code>trigger.EVENT_NOTIFY6</code>)
Scan completed	<code>trigger.EVENT_SCAN_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY8</code>)
Measure completed	<code>trigger.EVENT_SCAN_MEASURE_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY7</code>)
Limit value for scan reached	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code> (returns <code>trigger.EVENT_NOTIFY3</code>)

Example 1

```
scan.start.stimulus = trigger.EVENT_SCAN_CHANNEL_READY
```

Start the scan when the channels have closed.

Example 2

```
scan.start.stimulus = trigger.EVENT_NONE
```

Start the scan immediately.

Example 3

```
scan.start.stimulus = trigger.EVENT_DIGIO3
```

The scan begins when the instrument receives a signal from digital I/O line 3.

Also see

[Scanning and triggering](#) (on page 5-29)

scan.stepcount

This attribute returns the number of steps in the present scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
scanStepCount = scan.stepcount
```

<code>scanStepCount</code>	The present step count value

Details

This is set by the number of steps in the active scan list.

Example

```
print(scan.stepcount)
```

Responds with the present step count.
Output assuming there are five steps in the scan list:
5

Also see

[scan.add\(\)](#) (on page 14-374)
[scan.create\(\)](#) (on page 14-383)
[Scanning and triggering](#) (on page 5-29)

script.catalog()

This function returns an iterator that can be used in a `for` loop to iterate over all the scripts stored in nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in script.catalog() do body end
```

<i>name</i>	String representing the name of the script returned from <code>script.catalog()</code>
<i>body</i>	Code that implements the body of the <code>for</code> loop to process the return names from the <code>catalog</code> command

Details

This function accesses the catalog of scripts stored in nonvolatile memory, which allows you to process all scripts in nonvolatile memory. The entries are enumerated in no particular order.

Each time the body of the function executes, *name* takes on the name of one of the scripts stored in nonvolatile memory. The `for` loop repeats until all scripts have been iterated.

Example

<pre>for name in script.catalog() do print(name) end</pre>	Retrieve the catalog listing for user scripts. <code>print(name)</code> represents the body parameter shown in the Usage.

Also see

None

script.delete()

This function deletes a script from the runtime memory and nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.delete("scriptName")
```

scriptName	A string that represents the name of the script

Details

When a script is deleted, the global variable referring to this script is also deleted.

You must delete an existing script before you can use the name of that script again. Scripts are not automatically overwritten.

Example

script.delete("test8")	Deletes a user script named test8 from nonvolatile memory and the global variable named test8.

Also see

- [Deleting a user script using a remote interface](#) (on page 13-13)
- [scriptVar.save\(\)](#) (on page 14-408)

script.load()

This function creates a script from a specified file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.load("file")
scriptVar = script.load("file")
```

<i>file</i>	A string that contains the path and file name of the script file to load; if <i>scriptVar</i> is not defined, this name is used as the global variable name for this script
<i>scriptVar</i>	The created script; a global variable with this name is used to reference the script

Details

The name that is used for *scriptVar* must not already exist as a global variable. In addition, the *scriptVar* name must be a global reference and not a local variable, table, or array.

For external scripts, the root folder of the USB flash drive has the absolute path `/usb1/`.

Example

<pre>test8 = script.load("/usb1/testSetup.tsp")</pre>	Loads the script with the file name <code>testSetup.tsp</code> that is on the USB flash drive and names it <code>test8</code> .

Also see

None

scriptVar.run()

This function runs a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.run()  
scriptVar()
```

scriptVar	The name of the variable that references the script

Details

The `scriptVar.run()` function runs the script referenced by `scriptVar`. You can also run the script by using `scriptVar()`.

Example

test8.run()
Runs the script referenced by the variable <code>test8</code> .

Also see

None

scriptVar.save()

This function saves the script to nonvolatile memory or to a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.save()  
scriptVar.save("filename")
```

<i>scriptVar</i>	The name of variable that references the script
<i>filename</i>	A string that contains the file name to use when saving the script to a USB flash drive

Details

The `scriptVar.save()` function saves a script to nonvolatile memory or a USB flash drive. The root folder of the USB flash drive has the absolute path `/usb1/`.

If no *filename* is specified, the script is saved to internal nonvolatile memory. If a *filename* is given, the script is saved to the USB flash drive.

If you set *scriptVar* to `autoexec`, the script is run when the instrument powers up. You must delete the existing `autoexec` script before saving the new one. Note that performing a system reset does not delete the `autoexec` script.

You can add the file extension, but it is not required. The only allowed extension is `.tsp` (see Example 2).

Example 1

```
test8.save()
```

Saves the script referenced by the variable `test8` to nonvolatile memory.

Example 2

```
test8.save("/usb1/myScript.tsp")
```

Saves the script referenced by the variable `test8` to a file named `myScript.tsp` on your USB flash drive.

Also see

[Working with scripts](#) (on page 13-7)

scriptVar.source

This attribute contains the source code of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
code = scriptVar.source
```

<code>code</code>	The body of the script
<code>scriptVar</code>	The name of the variable that references the script that contains the source code

Details

The body of the script is a single string with lines separated by the new line character.

Example

```
print(test7.source)
```

Assuming a script named `test7` was created on the instrument, this example retrieves the source code.

Output:

```
reset()  
display.setText(display.TEXT1, "Text on line 1")  
display.setText(display.TEXT2, "Text on line 2")
```

Also see

[scriptVar.save\(\)](#) (on page 14-408)

slot[slot].amps.endchannel

This attribute indicates the last channel in the specified slot that supports amps measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].amps.endchannel
```

<code>value</code>	The channel number of the last channel in the group of channels that supports amps measurements
<code>slot</code>	Slot number

Details

The channels that support amps measurements are grouped, so you can use the start and end channel numbers to identify a group of channels that supports amps measurements. If the card supports amps measurements, the returned value is the number of the end channel. If only one channel on the card supports amps measurements, the end channel matches the start channel. If the channel does not support amps measurements, the return is `nil`.

Example

```
print(slot[2].amps.startchannel)
print(slot[2].amps.endchannel)
```

If the DAQ6510 contains a 7702 switching module in slot 2, these commands return 41 for the start channel and 42 for the end channel.

Also see

[slot\[slot\].amps.startchannel](#) (on page 14-410)

slot[slot].amps.startchannel

This attribute indicates the first channel in the specified slot that supports current measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].amps.startchannel
```

<i>value</i>	The channel number of the first channel in the group of channels that supports amps measurements
<i>slot</i>	Slot number

Details

The channels that support current measurements are grouped, so you can use the start and end channel numbers to identify the group of channels that supports current measurements. If the card supports current measurements, the returned value is the number of the start channel. If only one channel on the card supports current measurements, the start channel matches the end channel. If the channel does not support current measurements, the return is `nil`.

Example

```
print(slot[2].amps.startchannel)
print(slot[2].amps.endchannel)
```

If the DAQ6510 contains a 7702 switching module in slot 2, these commands return 41 for the start channel and 42 for the end channel.

Also see

[slot\[slot\].amps.endchannel](#) (on page 14-409)

slot[slot].analogoutput.endchannel

This attribute indicates the last channel in the specified slot that supports analog outputs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].analogoutput.endchannel
```

value	The channel number of the last channel in the group of channels that supports analog outputs
slot	Slot number

Details

The channels that support analog outputs are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card supports analog outputs, the returned value is the number of the last channel. If only one channel on the card supports analog outputs, the start channel matches the end channel. If the channel does not support analog outputs, the return is `nil`.

Example

```
print(slot[1].analogoutput.startchannel)
print(slot[1].analogoutput.endchannel)
```

If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 23 for the start channel and 24 for the end channel.

Also see

[slot\[slot\].analogoutput.startchannel](#) (on page 14-411)

slot[slot].analogoutput.startchannel

This attribute indicates the first channel in the specified slot that supports analog outputs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].analogoutput.startchannel
```

value	The channel number of the first channel in the group of channels that supports analog outputs
slot	Slot number

Details

The channels that support analog outputs are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card supports analog outputs, the returned value is the number of the start channel. If only one channel on the card supports analog outputs, the start channel matches the end channel. If the channel does not support analog outputs, the return is `nil`.

Example

```
print(slot[1].analogoutput.startchannel)
print(slot[1].analogoutput.endchannel)
```

If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 23 for the start channel and 24 for the end channel.

Also see

[slot\[slot\].analogoutput.endchannel](#) (on page 14-411)

slot[slot].commonsideohms

This attribute indicates whether a card in the specified slot supports commonside channels for 4-wire resistance measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
commonsideohms = slot[slot].commonsideohms
```

<code>commonsideohms</code>	Indication of whether or not commonside 4-wire ohm channels are supported
<code>slot</code>	Slot number

Details

If commonside 4-wire resistance channels are supported, the returned value is 1.
If commonside 4-wire resistance channels are not supported, the return value is `nil`.

Example

```
print(slot[1].commonsideohms)
```

Query if slot 1 supports commonside 4-wire ohms channels.

Also see

[channel.setcommonsideohms\(\)](#) (on page 14-84)

slot[slot].digitalio.endchannel

This attribute indicates the last channel in the specified slot that supports digital inputs and outputs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].digitalio.endchannel
```

value	The channel number of the last channel in the group of channels that supports digital inputs and outputs
slot	Slot number

Details

The channels that support digital inputs and outputs are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card supports digital inputs and outputs, the returned value is the number of the end channel. If only one channel on the card supports digital inputs and outputs, the start channel matches the end channel. If the channel does not support digital inputs and outputs, the return is `nil`.

Example

```
print(slot[1].digitalio.startchannel)
print(slot[1].digitalio.endchannel)
```

If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 21 for the start channel and 22 for the end channel.

Also see

[slot\[slot\].digitalio.startchannel](#) (on page 14-413)

slot[slot].digitalio.startchannel

This attribute indicates the first channel in the specified slot that supports digital inputs and outputs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].digitalio.startchannel
```

value	The channel number of the first channel in the group of channels that supports digital inputs and outputs
slot	Slot number

Details

The channels that support digital inputs and outputs are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card supports digital inputs and outputs, the returned value is the number of the start channel. If only one channel on the card supports digital inputs and outputs, the start channel matches the end channel. If the channel does not support digital inputs and outputs, the return is `nil`.

Example

```
print(slot[1].digitalio.startchannel)
print(slot[1].digitalio.endchannel)
```

If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 21 for the start channel and 22 for the end channel.

Also see

[slot\[slot\].digitalio.endchannel](#) (on page 14-413)

slot[slot].idn

This attribute returns a string that contains information about the switching module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
idnString = slot[slot].idn
```

<i>idnString</i>	The return string
<i>slot</i>	Slot number (1 or 2)

Details

The information that is returned depends on whether the switching module in the slot is a physical card or pseudocard.

For physical cards, this returns a comma-separated string that contains the model number, description, firmware revision, and serial number of the switching module installed in the specified slot.

For pseudocards, the response is `Pseudo`, followed by the model number, description, and `???` for the firmware revision and serial number.

Example

```
print(slot[1].idn)
```

If a 7706 module is installed in slot 1, the response is:

```
7706,25Ch ALL-IN-ONE w/DIO,01.40e,1243657
```

Also see

None

slot[slot].isolated.endchannel

This attribute indicates the last channel in the specified slot that has isolated channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].isolated.startchannel
```

value	The channel number of the first channel in the group that is isolated
slot	Slot number

Details

The channels that are isolated are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card includes isolated channels, the returned value is the number of the end channel. If only one channel on the card is isolated, the start channel matches the end channel. If none of the channels are isolated, the return is `nil`.

This command is not available for the Model 7709 switching module.

Example

```
print(slot[2].isolated.startchannel)
print(slot[2].isolated.endchannel)
```

If the DAQ6510 contains a 7705 switching module in slot 2, these commands return 1 for the start channel and 40 for the end channel.

Also see

[slot\[slot\].isolated.startchannel](#) (on page 14-416)

slot[slot].isolated.startchannel

This attribute indicates the first channel in the specified slot that has isolated channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].isolated.startchannel
```

value	The channel number of the first channel in the group of channels that are isolated
slot	Slot number

Details

The channels that are isolated are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card includes isolated channels, the returned value is the number of the first isolated channel. If only one channel on the module is isolated, the start channel matches the end channel. If none of the channels are isolated, the return is `nil`.

This command is not available for the Model 7709 switching module.

Example

```
print(slot[2].isolated.startchannel)
print(slot[2].isolated.endchannel)
```

If the DAQ6510 contains a 7705 switching module in slot 2, these commands return 1 for the start channel and 40 for the end channel.

Also see

[slot\[slot\].isolated.endchannel](#) (on page 14-416)

slot[slot].matrix.columns

This attribute returns the number of columns in the matrix for the card in the specified slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].matrix.columns
```

value	The number of columns in the matrix
slot	The slot number

Details

This attribute is only available if the installed switching module supports matrix channels. If matrix channels are not available, the return value is `nil`.

Example

```
print(slot[2].matrix.columns)
```

Returns the number of columns in the matrix on the switching module in slot 2.
Example output if the switching module is a 7709:
8

Also see

[slot\[slot\].matrix.rows](#) (on page 14-418)

slot[slot].matrix.rows

This attribute returns the number of rows in the matrix on the card in the specified slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].matrix.rows
```

value	Number of rows in the matrix card of the selected slot
slot	Slot number (1 or 2)

Details

This attribute is only available for a slot if the installed card supports matrix channels. If matrix channels are not available, the return value is `nil`.

Example

<pre>print(slot[2].matrix.rows)</pre>	Returns the number of rows in the matrix on the switching module in slot 2. Example output: 6

Also see

- [slot\[slot\].matrix.columns](#) (on page 14-417)
- [slot\[slot\].idn](#) (on page 14-415)

slot[slot].maxvoltage

This attribute returns the maximum voltage of all channels on a switching module in the specified slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
maximumVolts = slot[slot].maxvoltage
```

<i>maximumVolts</i>	The maximum voltage
<i>slot</i>	Slot number (1 or 2)

Details

This command is only available for a slot if the installed switching module supports voltage settings.

This command is not available for the 7709 switching module.

Example

```
maxVolts2 = slot[2].maxvoltage  
print(maxVolts2)
```

Query the maximum voltage on slot 2. The output is similar to:
3.0000000000e+02

Also see

[slot\[slot\].idn](#) (on page 14-415)

slot[slot].pseudocard

This attribute specifies a pseudocard to implement for the designated slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	slot.PSEUDO_NONE

Usage

```
pseudoCard = slot[slot].pseudocard
slot[slot].pseudocard = pseudoCard
```

pseudoCard	The pseudocards available for the DAQ6510 are: - slot.PSEUDO_NONE for no pseudocard 7700: 20-channel Differential Multiplexer Module 7701: 32-channel Differential Multiplexer Module 7702: 40-channel Differential Multiplexer Module with Screw Terminals 7703: 32-channel High Speed Differential Multiplexer Module 7705: 40-channel Single-pole Control Module 7706: 20-channel All-in-One I/O Module 7707: 32-channel Digital I/O Module with 10-channel Differential Multiplexer 7708: 40-channel Differential Multiplexer Module with Automatic CJC and Screw Terminals 7709: 6 × 8 Matrix Module 7710: 20-channel Solid-state Differential Multiplexer with Automatic CJC 7711: 2 GHz 50 Ohm RF Module 7712: 3.5 GHz 50 Ohm RF Module
slot	Slot number (1 or 2)

Details

Pseudocards allow you to configure your system without having an actual switching module installed in your system. You can perform open, close, and scan operations and configure your system with pseudocards.

This command is only applicable to a slot that does not have a switching module or pseudocard installed. If a pseudocard is presently assigned to the slot, you must set the slot to no pseudocard before assigning the new pseudocard.

After assigning a pseudocard, you can use valid commands for the switching module for that slot.

Changing the pseudocard assignment from a pseudocard to no pseudocard invalidates scan lists that include that slot.

If a module is installed in the slot, this command returns `nil`. If the slot is empty and no pseudocard is installed, the return is `0`.

Example

```
slot[1].pseudocard = slot.PSEUDO_NONE
slot[1].pseudocard = 7702
print(slot[1].idn)
```

Sets slot 1 to no pseudocard, then sets slot 1 to simulate a 7702 switching module. Output is similar to:

```
7702, Pseudo 40Ch
Mux, ???????, ???????????
```

Also see

- [Pseudocards](#) (on page 5-2)
- [slot\[slot\].idn](#) (on page 14-415)

slot[slot].tempsensor

This attribute indicates if the module in the specified slot supports temperature sensor channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].tempsensor
```

value	If temperature sensor channels are supported, returns 1
slot	Slot number

Details

This command is only available for a slot if the installed module supports temperature sensor channels. If temperature sensor channels are not available, the return value is `nil`.

Example

```
TempSensor = slot[1].tempsensor
print(TempSensor)
```

Query to determine if slot 1 supports temperature sensor channels. If it does support temperature sensor channels, the output is:
1

Also see

[slot\[slot\].idn](#) (on page 14-415)

slot[slot].totalizer.endchannel

This attribute indicates the last totalizer channel in the specified slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].totalizer.endchannel
```

value	The channel number of the last totalizer channel in the group
slot	Slot number

Details

The totalizer channels are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card includes totalizer channels, the returned value is the number of the last totalizer channel. If there is only one totalizer channel, the start channel matches the end channel. If none of the channels are totalizers, the return is `nil`.

Example

```
print(slot[1].totalizer.startchannel)
print(slot[1].totalizer.endchannel)
```

If the DAQ6510 contains a 7706 card in slot 1, these commands return 25 for both the start channel and end channel.

Also see

[slot\[slot\].totalizer.startchannel](#) (on page 14-422)

slot[slot].totalizer.startchannel

This attribute indicates the first totalizer channel in the specified slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].totalizer.startchannel
```

value	The channel number of the first totalizer channel in the group
slot	Slot number

Details

The totalizer channels are grouped, so you can use the start and end channel numbers to identify the group of channels. If the card includes totalizer channels, the returned value is the number of the first totalizer channel. If there is only one totalizer channel, the start channel matches the end channel. If none of the channels are totalizers, the return is `nil`.

Example

```
print(slot[1].totalizer.startchannel)
print(slot[1].totalizer.endchannel)
```

If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 25 for both the start channel and end channel.

Also see

[slot\[slot\].totalizer.endchannel](#) (on page 14-421)

slot[slot].voltage.endchannel

This attribute indicates the last channel in the specified slot that supports voltage or 2-wire measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].voltage.endchannel
```

value	The channel number of the last channel in the group of channels that supports voltage or two-wire measurements
slot	Slot number

Details

The channels that support voltage or 2-wire measurements are grouped, so you can use the start and end channel numbers to identify the group. If the card supports voltage or 2-wire measurements, the returned value is the number of the last channel. If only one channel on the card supports voltage or 2-wire measurements, the start channel matches the end channel. If the channel does not support voltage or 2-wire measurements, the return is `nil`.

This command is not available for the 7709 switching module.

Example

```
print(slot[1].voltage.startchannel)
print(slot[1].voltage.endchannel)
```

If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 1 for the start channel and 20 for the end channel.

Also see

[slot\[slot\].voltage.startchannel](#) (on page 14-423)

slot[slot].voltage.startchannel

This attribute indicates the first channel in the specified slot that supports voltage or 2-wire measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[slot].voltage.startchannel
```

value	The channel number of the first channel in the group of channels that supports voltage or two-wire measurements
slot	Slot number

Details

The channels that support voltage or 2-wire measurements are grouped, so you can use the start and end channel numbers to identify the group. If the card supports voltage or 2-wire measurements, the returned value is the number of the start channel. If only one channel on the card supports voltage or 2-wire measurements, the start channel matches the end channel. If the channel does not support voltage or 2-wire measurements, the return is `nil`.

This command is not available for the 7709 switching module.

Example

```
print(slot[1].voltage.startchannel)
print(slot[1].voltage.endchannel)
```

If the DAQ6510 contains a 7706 switching module in slot 1, these commands return 1 for the start channel and 20 for the end channel.

Also see

[slot\[slot\].voltage.endchannel](#) (on page 14-423)

status.clear()

This function clears event registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.clear()
```

Details

This command clears the event registers of the Questionable Event and Operation Event Register set. It does not affect the Questionable Event Enable or Operation Event Enable registers.

Example

<code>status.clear()</code>	Clear the bits in the registers

Also see

[*CLS](#) (on page 15-2)

status.condition

This attribute stores the status byte condition register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
statusByte = status.condition
```

<code>statusByte</code>	The status byte

Details

You can use this command to read the status byte, which is returned as a numeric value.

When an enabled status event occurs, a summary bit is set in this register to indicate the event occurrence. The returned value can indicate that one or more status events occurred. If more than one bit of the register is set, *statusByte* equals the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128). See [Understanding bit settings](#) (on page 16-18) for additional information about reading bit values.

NOTE

If you are using the GPIB, USB, or VXI-11 serial poll sequence of the DAQ6510 to get the status byte (also called a serial poll byte), B6 is the Request for Service (RQS) bit. If the bit is set, it indicates that a serial poll (SRQ) has occurred. For additional detail, see [Serial polling and SRQ](#) (on page 16-15).

The meanings of the individual bits of this register are shown in the following table.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	<code>status.MSB</code>	An enabled measurement event
1	2	Not used	
2	4	<code>status.EAV</code>	An error or status message is present in the Error Queue
3	8	<code>status.QSB</code>	An enabled questionable event
4	16	<code>status.MAV</code>	A response message is present in the Output Queue
5	32	<code>status.ESB</code>	An enabled standard event
6	64	<code>status.MSS</code>	An enabled summary bit of the status byte register is set
7	128	<code>status.OSB</code>	An enabled operation event

Example

```
statusByte = status.condition
print(statusByte)
```

Returns `statusByte`.

Example output:

```
1.29000e+02
```

Converting this output (129) to its binary equivalent yields 1000 0001

Therefore, this output indicates that the set bits of the status byte condition register are presently B0 (MSS) and B7 (OSB).

Also see

None

status.operation.condition

This attribute reads the Operation Event Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
operationRegister = status.operation.condition
```

<code>operationRegister</code>	The status of the Operation Status Register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This command reads the contents of the Operation Condition Register, which is one of the Operation Event Registers.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-18).

Example

<pre>print(status.operation.condition)</pre>	Returns the contents of the register.

Also see

[Operation Event Register](#) (on page 16-8)

status.operation.enable

This attribute sets or reads the contents of the Operation Event Enable Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
operationRegister = status.operation.enable
status.operation.enable = operationRegister
```

<i>operationRegister</i>	The status of the Operation Status Register

Details

This command sets or reads the contents of the Enable register of the Operation Event Register.

When one of these bits is set, when the corresponding bit in the Operation Event Register or Operation Condition Register is set, the OSB bit in the Status Byte Register is set.

Example

<pre>-- decimal 20480 = binary 0101 0000 0000 0000 operationRegister = 20480 status.operation.enable = operationRegister</pre>	Sets the 12 and 14 bits of the Operation Status Enable register using a decimal value.

Also see

- [Operation Event Register](#) (on page 16-8)
- [Understanding bit settings](#) (on page 16-18)

status.operation.event

This attribute reads the Operation Event Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
operationRegister = status.operation.event
```

operationRegister	The status of the Operation Status Register

Details

This attribute reads the operation event register of the status model.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Example

```
status.request_enable = status.OSB
status.operation.setmap(0, 4918, 4917)
status.operation.enable = 1
defbuffer1.clear()
defbuffer1.fillmode = buffer.FILL_ONCE
defbuffer1.capacity = 10
dmm.measure.count = 10
dmm.measure.read()
print(status.operation.event)
```

Set bits in the Status Request Enable Register to record an enabled event in the Operation Status Register. Map event number 4918 (a default buffer is full) to set bit 0 in the Operation Event Register and event number 4917 (a default buffer is empty) to clear bit 0.

Clear defbuffer1.

Set defbuffer1 to fill once.

Resizes defbuffer1 to 10 readings.

Sets the measure count to 10 readings and makes a measurement.

Reads the operation event register.

Output:

1

Also see

[Operation Event Register](#) (on page 16-8)

status.operation.getmap()

This function requests the mapped set event and mapped clear event status for a bit in the Operation Event Registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

setEvent, clearEvent = status.operation.getmap(*bitNumber*)

<i>setEvent</i>	The event mapped to set this bit; 0 if no mapping
<i>clearEvent</i>	The event mapped to clear this bit; 0 if no mapping
<i>bitNumber</i>	The bit number to check

Details

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

<pre>print(status.operation.getmap(0))</pre>	Query bit 0 of the Operation Event Register. Example output: 4918 4917

Also see

- [Operation Event Register](#) (on page 16-8)
- [Programmable status register sets](#) (on page 16-5)
- [status.operation.setmap\(\)](#) (on page 14-430)

status.operation.setmap()

This function allows you to map events to bits in the Operation Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.operation.setmap(bitNumber, setEvent)
status.operation.setmap(bitNumber, setEvent, clearEvent)
```

<i>bitNumber</i>	The bit number that is mapped to an event: 0 to 14
<i>setEvent</i>	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
<i>clearEvent</i>	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

Example

```
status.operation.setmap(0, 2731, 2732)
```

When event 2731 (trigger model initiated) occurs, bit 0 in the condition and event registers of the Operation Event Register are set. When event 2732 (trigger model idled) occurs, bit 0 in the condition register is cleared.

Also see

- [Operation Event Register](#) (on page 16-8)
- [Programmable status register sets](#) (on page 16-5)
- [status.operation.getmap\(\)](#) (on page 14-429)

status.preset()

This function resets all bits in the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.preset()
```

Details

This function clears the event registers and the enable registers for operation and questionable. It will not clear the Service Request Enable Register (*SRE) to Standard Request Enable Register (*ESE).

Preset does not affect the event queue.

The Standard Event Status Register is not affected by this command.

Example

<pre>status.preset()</pre>	Resets the instrument status model.

Also see

[Status model](#) (on page 16-1)

status.questionable.condition

This attribute reads the Questionable Condition Register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
questionableRegister = status.questionable.condition
```

<pre>questionableRegister</pre>	The value of the register (0 to 65535)

Details

This command reads the contents of the Questionable Condition Register, which is one of the Questionable Event Registers.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-18).

Example

<pre>print(status.questionable.condition)</pre>	Reads the Questionable Condition Register.

Also see

[Questionable Event Register](#) (on page 16-7)

[Understanding bit settings](#) (on page 16-18)

status.questionable.enable

This attribute sets or reads the contents of the questionable event enable register of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
questionableRegister = status.questionable.enable
status.questionable.enable = questionableRegister
```

<i>questionableRegister</i>	The value of the register (0 to 65535)

Details

This command sets or reads the contents of the Enable register of the Questionable Event Register.

When one of these bits is set, when the corresponding bit in the Questionable Event Register or Questionable Condition Register is set, the MSB and QSM bits in the Status Byte Register are set.

For detail on interpreting the value of a register, see [Understanding bit settings](#) (on page 16-18).

Example

```
status.questionable.enable = 17
print(status.questionable.enable)
```

Set bits 0 and 4 of the Questionable Event Enable Register.
Returns 17, which indicates the register was set correctly.

Also see

[Questionable Event Register](#) (on page 16-7)

status.questionable.event

This attribute reads the Questionable Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
questionableRegister = status.questionable.event
```

<i>questionableRegister</i>	The value of the questionable status register (0 to 65535)

Details

This query reads the contents of the questionable status event register. After sending this command and addressing the instrument to talk, a value is sent to the computer. This value indicates which bits in the appropriate register are set.

The Questionable Register can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set the Questionable Register to the sum of their decimal weights. For example, to set bits B12 and B13, set the Questionable Register to 12,288 (which is the sum of 4,096 + 8,192).

Example 1

```
-- decimal 66 = binary 0100 0010
questionableRegister = 66
status.questionable.enable = questionableRegister
```

Uses a decimal value to set bits B1 and B6 of the status questionable enable register.

Example 2

```
-- decimal 2560 = binary 00001010 0000 0000
questionableRegister = 2560
status.questionable.enable = questionableRegister
```

Uses a decimal value to set bits B9 and B11 of the status questionable enable register.

Also see

[Questionable Event Register](#) (on page 16-7)

status.questionable.getmap()

This function requests the mapped set event and mapped clear event status for a bit in the Questionable Event Registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setEvent, clearEvent = status.questionable.getmap(bitNumber)
```

setEvent	The event mapped to set this bit; 0 if no mapping
clearEvent	The event mapped to clear this bit; 0 if no mapping
bitNumber	The bit number to check (0 to 14)

Details

When you query the mapping for a specific bit, the instrument returns the events that were mapped to set and clear that bit. Zero (0) indicates that the bits have not been set.

Example

```
print(status.questionable.getmap(9))
```

Returns the events that were mapped to set and clear bit 9.

Also see

[Questionable Event Register](#) (on page 16-7)
[status.questionable.setmap\(\)](#) (on page 14-434)

status.questionable.setmap()

This function maps events to bits in the questionable event registers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.questionable.setmap(bitNumber, setEvent)
status.questionable.setmap(bitNumber, setEvent, clearEvent)
```

<i>bitNumber</i>	The bit number that is mapped to an event: 0 to 14
<i>setEvent</i>	The number of the event that sets the bits in the condition and event registers; 0 if no mapping
<i>clearEvent</i>	The number of the event that clears the bit in the condition register; 0 if no mapping

Details

You can map events to bits in the event registers with this command. This allows you to cause bits in the condition and event registers to be set or cleared when the specified events occur. You can use any valid event number as the event that sets or clears bits.

When a mapped event is programmed to set bits, the corresponding bits in both the condition register and event register are set when the event is detected.

When a mapped event is programmed to clear bits, the bit in the condition register is set to 0 when the event is detected.

If the event is set to zero (0), the bit is never set.

Example

```
status.questionable.setmap(0, 4917, 4918)
```

When event 4917 (a default buffer is 0% filled) occurs, bit 0 is set in the condition register and the event register of the Questionable Event Register. When event 4918 (a default buffer is 100% filled) occurs, bit 0 in the condition register is cleared.

Also see

[status.questionable.getmap\(\)](#) (on page 14-433)

status.request_enable

This attribute stores the settings of the Service Request (SRQ) Enable Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
SRQEnableRegister = status.request_enable
status.request_enable = SRQEnableRegister
```

SRQEnableRegister

The status of the service request (SRQ) enable register:

- No bits set: 0
- Other bit settings: 0 to 255
- Send to clear all bits: 0

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	status.MSB	An enabled event in the Measurement Event Register has occurred.
1	2	Not used	Not used.
2	4	status.EAV	An error or status message is present in the Error Queue.
3	8	status.QSB	An enabled event in the Questionable Status Register has occurred.
4	16	status.MAV	A response message is present in the Output Queue.
5	32	status.ESB	An enabled event in the Standard Event Status Register has occurred.
6	64	Not used	Not used.
7	128	status.OSB	An enabled event in the Operation Status Register has occurred.

Example 1

```
requestSRQEnableRegister = status.MSB + status.OSB
status.request_enable = requestSRQEnableRegister
```

Uses constants to set the MSB and OSB bits of the service request (SRQ) enable register and clear all other bits.

Example 2

```
-- decimal 129 = binary 10000001
requestSRQEnableRegister = 129
status.request_enable = requestSRQEnableRegister
```

Uses a decimal value to set the MSB and OSB bits and clear all other bits of the service request (SRQ) enable register.

Example 3

```
status.request_enable = 0
```

Clear the register.

Also see

- [Status model](#) (on page 16-1)
- [Understanding bit settings](#) (on page 16-18)

status.standard.enable

This attribute reads or sets the bits in the Status Enable register of the Standard Event Register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	status.preset()	Not applicable	0

Usage

```
standardRegister = status.standard.enable
status.standard.enable = standardRegister
```

<i>standardRegister</i>	The value of the Status Enable register of the Standard Event Register: 0 to 255

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte Register is set to on.

To set a bit on, send the constant or value of the bit as the *standardRegister* parameter.

You can set the bit as a constant or a numeric value, as shown in the table below. To set more than one bit of the register, you can send multiple constants with + between them. You can also set *standardRegister* to the sum of their decimal weights. For example, to set bits B0 and B2, set *standardRegister* to 5 (which is the sum of 1 + 4). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.QYE
```

When zero (0) is returned, no bits are set. You can also send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	<code>status.standard.OPC</code>	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-9) command or TSP opc() (on page 14-367) function.
1	2	Not used	Not used.
2	4	<code>status.standard.QYE</code>	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	<code>status.standard.PON</code>	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example 1

```
standardRegister = status.standard.OPC + status.standard.QYE
status.standard.enable = standardRegister
```

Uses constants to set the OPC and QYE bits of the standard event status enable register.

Example 2

```
-- decimal 5 = binary 0000 0101
standardRegister = 5
status.standard.enable = standardRegister
```

Uses a decimal value to set the OPC and QYE bits of the standard event status enable register.

Also see

[Standard Event Register](#) (on page 16-3)

[Understanding bit settings](#) (on page 16-18)

status.standard.event

This attribute returns the contents of the Standard Event Status Register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	status.preset()	Not applicable	0

Usage

```
standardRegister = status.standard.event
```

<i>standardRegister</i>	The status of the standard event status register

Details

When this command returns zero (0), no bits are set. You can send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.standard.OPC	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-9) command or TSP opc() (on page 14-367) function.
1	2	Not used	Not used.
2	4	status.standard.QYE	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	status.standard.PON	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

```
print(status.standard.event)
```

May return the value 129, showing that the Standard Event Status Register contains binary 10000001

Also see

[Standard Event Register](#) (on page 16-3)

[Understanding bit settings](#) (on page 16-18)

timer.cleartime()

This function resets the timer to zero (0) seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
timer.cleartime()
```

Example

```
dataqueue.clear()
dataqueue.add(35)
timer.cleartime()
delay(0.5)
dt = timer.gettime()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it.

Output:

```
Delay time was 0.500099
35
```

Also see

[timer.gettime\(\)](#) (on page 14-440)

timer.gettime()

This function measures the elapsed time since the timer was last cleared.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
time = timer.gettime()
```

<i>time</i>	The elapsed time in seconds (1 μ s resolution)

Example

```
dataqueue.clear()
dataqueue.add(35)
timer.cleartime()
delay(0.5)
dt = timer.gettime()
print("Delay time was " .. dt)
print(dataqueue.next())
```

Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it.

Output:

```
Delay time was 0.500099
35
```

Also see

[timer.cleartime\(\)](#) (on page 14-439)

trigger.blender[N].clear()

This function clears the blender event detector and resets the overrun indicator of blender *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].clear()
```

<i>N</i>	The blender number: up to two

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector.

Example

```
trigger.blender[2].clear()
```

Clears the event detector for blender 2.

Also see

None

trigger.blender[N].orenable

This attribute selects whether the blender performs OR operations or AND operations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger blender N reset	Configuration script	false (AND)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

<i>orenable</i>	The type of operation: - OR operation: true - AND operation: false
<i>N</i>	The blender number: up to two

Details

This command selects whether the blender waits for any one event (OR) or waits for all selected events (AND) before signaling an output event.

Example

```
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.
```

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 14-443)

trigger.blender[N].overrun

This attribute indicates whether or not an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Trigger blender <i>N</i> clear Trigger blender <i>N</i> reset	Not applicable	Not applicable

Usage

```
overrun = trigger.blender[N].overrun
```

<i>overrun</i>	Trigger blender overrun state: <code>true</code> or <code>false</code>
<i>N</i>	The blender number: up to two

Details

Indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector that is built into the event blender itself.

This command does not indicate if an overrun occurred in any other part of the trigger model or in any other trigger object that is monitoring the event. It also is not an indication of an action overrun.

Example

```
print(trigger.blender[1].overrun)
```

If an event was ignored, the output is `true`.

If an event was not ignored, the output is `false`.

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 14-443)

trigger.blender[N].reset()

This function resets some of the trigger blender settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].reset()
```

<i>N</i>	The trigger event blender: up to two

Details

The `trigger.blender[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.blender[N].orenable`
- `trigger.blender[N].stimulus[M]`

It also clears `trigger.blender[N].overrun`.

Example

<pre>trigger.blender[1].reset()</pre>
Resets the trigger blender 1 settings to factory defaults.

Also see

- [trigger.blender\[N\].orenable](#) (on page 14-441)
- [trigger.blender\[N\].overrun](#) (on page 14-442)
- [trigger.blender\[N\].stimulus\[M\]](#) (on page 14-444)

trigger.blender[N].stimulus[M]

This attribute specifies the events that trigger the blender.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger blender N reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.blender[N].stimulus[M]  
trigger.blender[N].stimulus[M] = event
```

event	The event that triggers the blender action; see Details
N	An integer that represents the trigger event blender (up to two)
M	An integer representing the stimulus index (1 to 4)

Details

There are four stimulus inputs that can each select a different event.
Use none to disable the blender input.

The *event* parameter may be any of the trigger events shown in the following table.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	<code>trigger.EVENT_NONE</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
A command interface trigger (bus trigger): - Any remote interface: <code>*TRG</code> - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Scan alarm limit exceeded	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Channel closed	<code>trigger.EVENT_SCAN_CHANNEL_READY</code> (returns <code>trigger.EVENT_NOTIFY6</code>)
Scan completed	<code>trigger.EVENT_SCAN_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY8</code>)
Measure completed	<code>trigger.EVENT_SCAN_MEASURE_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY7</code>)
Limit value for scan reached	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code> (returns <code>trigger.EVENT_NOTIFY3</code>)

Example

```

digio.line[3].mode = digio.MODE_TRIGGER_IN
digio.line[5].mode = digio.MODE_TRIGGER_IN
trigger.digin[3].edge = trigger.EDGE_FALLING
trigger.digin[5].edge = trigger.EDGE_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

```

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 14-443)

trigger.blender[N].wait()

This function waits for a blender trigger event to occur.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.blender[N].wait(timeout)
```

<i>triggered</i>	Trigger detection indication for blender
<i>N</i>	The trigger blender (up to two) on which to wait
<i>timeout</i>	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_IN
digio.line[5].mode = digio.MODE_TRIGGER_IN
trigger.digin[3].edge = trigger.EDGE_FALLING
trigger.digin[5].edge = trigger.EDGE_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = trigger.EVENT_DIGIO3
trigger.blender[1].stimulus[2] = trigger.EVENT_DIGIO5
print(trigger.blender[1].wait(3))
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.
Wait 3 s while checking if trigger blender 1 event has occurred.

Also see

[trigger.blender\[N\].clear\(\)](#) (on page 14-440)

trigger.clear()

This function clears any pending command triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
trigger.clear()
```

Details

A command trigger indicates if a trigger event has been detected over a command interface since the last `trigger.wait()` command was sent. Command triggers are generated by:

- Sending `*TRG` over a remote interface
- GPIB `GET` bus commands
- USBTMC trigger messages
- VXI-11 device trigger commands

`trigger.clear()` clears the command triggers and discards the history of trigger events.

Example

<pre>*TRG print(trigger.wait(1)) trigger.clear() print(trigger.wait(1))</pre>	Generate a trigger event. Check if there are any pending trigger events. Output: <code>true</code> Clear any pending command triggers. Check if there are any pending trigger events. Output: <code>false</code>

Also see

[trigger.wait\(\)](#) (on page 14-542)

trigger.continuous

This attribute determines the trigger mode setting after bootup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Never	Nonvolatile memory	trigger.CONT_AUTO

Usage

```
setting = trigger.continuous
trigger.continuous = setting
```

<i>setting</i>	Do not start continuous measurements after boot: <code>trigger.CONT_OFF</code> Start continuous measurements after boot: <code>trigger.CONT_AUTO</code> Place the instrument into local control and start continuous measurements after boot: <code>trigger.CONT_RESTART</code>

Details

Conditions must be valid before continuous measurements can start.

When the restart parameter is selected, the instrument is placed in local mode, aborts any running scripts, and aborts any trigger models that are running. If the command is in a script, it is the last command that runs before the script is aborted. The restart parameter is not stored in nonvolatile memory, so it does not affect start up behavior.

The off and automatic parameters are stored in nonvolatile memory, so they affect start up behavior.

Example

```
trigger.continuous = trigger.CONT_OFF
```

When the instrument starts up, the measurement method is set to idle.

Also see

None

trigger.digin[N].clear()

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function clears the trigger event on a digital input line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.digin[N].clear()
```

<i>N</i>	Digital I/O trigger line: 1 to 6

Details

The event detector of a trigger enters the detected state when an event is detected. For the specified trigger line, this command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to `false`).

Example

```
trigger.digin[2].clear()
```

Clears the trigger event detector on I/O line 2.

Also see

[digio.line\[N\].mode](#) (on page 14-112)
[Digital I/O port configuration](#) (on page 8-3)
[trigger.digin\[N\].overrun](#) (on page 14-450)
[trigger.digin\[N\].wait\(\)](#) (on page 14-451)

trigger.digin[N].edge

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This attribute sets the edge used by the trigger event detector on the given trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EDGE_FALLING

Usage

```
detectedEdge = trigger.digin[N].edge
trigger.digin[N].edge = detectedEdge
```

<i>detectedEdge</i>	The trigger logic value: ■ Detect falling-edge triggers as inputs: <code>trigger.EDGE_FALLING</code> ■ Detect rising-edge triggers as inputs: <code>trigger.EDGE_RISING</code> ■ Detect either falling or rising-edge triggers as inputs: <code>trigger.EDGE_EITHER</code> ■ See Details for descriptions of values
<i>N</i>	Digital I/O trigger line: 1 to 6

Details

This command sets the logic on which the trigger event detector and the output trigger generator operate on the specified trigger line.

To directly control the line state, set the mode of the line to digital and use the write command. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse.

Trigger mode values

Value	Description
trigger.EDGE_FALLING	Detects falling-edge triggers as input when the line is configured as an input or open drain
trigger.EDGE_RISING	Detects rising-edge triggers as input when the line is configured as an open drain
trigger.EDGE_EITHER	Detects rising- or falling-edge triggers as input when the line is configured as an input or open drain

Example

<pre>digio.line[4].mode = digio.MODE_TRIGGER_IN trigger.digin[4].edge = trigger.EDGE_RISING</pre>	Sets the trigger mode for digital I/O line 4 to detect a rising-edge trigger as an input.

Also see

[digio.line\[N\].mode](#) (on page 14-112)
[digio.line\[N\].reset\(\)](#) (on page 14-114)
[digio.writeport\(\)](#) (on page 14-117)
[Digital I/O port configuration](#) (on page 8-3)
[trigger.digin\[N\].clear\(\)](#) (on page 14-448)

trigger.digin[N].overrun

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This attribute returns the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Digital I/O trigger <i>N</i> clear Digital I/O trigger <i>N</i> reset	Not applicable	Not applicable

Usage

```
overrun = trigger.digin[N].overrun
```

<i>overrun</i>	Trigger overrun state: true or false
<i>N</i>	Digital I/O trigger line: 1 to 6

Details

If this is `true`, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

```
overrun = trigger.digin[1].overrun
print(overrun)
```

If there is no trigger overrun on digital input 1, the output is:
`false`

Also see

- [digio.line\[N\].mode](#) (on page 14-112)
- [digio.line\[N\].reset\(\)](#) (on page 14-114)
- [Digital I/O port configuration](#) (on page 8-3)
- [trigger.digin\[N\].clear\(\)](#) (on page 14-448)

trigger.digin[N].wait()

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.digin[N].wait(timeout)
```

<i>triggered</i>	Trigger detected: <code>true</code> No triggers detected during the timeout period: <code>false</code>
<i>N</i>	Digital I/O trigger line: 1 to 6
<i>timeout</i>	Timeout in seconds

Details

This function pauses for up to *timeout* seconds for an input trigger. If one or more trigger events are detected since the last time `trigger.digin[N].wait()` or `trigger.digin[N].clear()` was called, this function returns a value immediately. After waiting for a trigger with this function, the event detector is automatically reset and is ready to detect the next trigger. This is true regardless of the number of events detected.

Example

<pre>digio.line[4].mode = digio.MODE_TRIGGER_IN triggered = trigger.digin[4].wait(3) print(triggered)</pre>	Waits up to 3 s for a trigger to be detected on trigger line 4, then outputs the results. Output if no trigger is detected: false Output if a trigger is detected: true

Also see

- [digio.line\[N\].mode](#) (on page 14-112)
- [Digital I/O port configuration](#) (on page 8-3)
- [trigger.digin\[N\].clear\(\)](#) (on page 14-448)

trigger.digout[N].assert()

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function asserts a trigger pulse on one of the digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

<pre>trigger.digout[N].assert()</pre>	
<i>N</i>	Digital I/O trigger line: 1 to 6

Details

Initiates a trigger event and does not wait for completion. The pulse width that is set determines how long the instrument asserts the trigger.

Example

<pre>digio.line[2].mode = digio.MODE_TRIGGER_OUT trigger.digout[2].pulsewidth = 20e-6 trigger.digout[2].assert()</pre>	Asserts a trigger on digital I/O line 2 with a pulse width of 20 μs.

Also see

- [digio.line\[N\].mode](#) (on page 14-112)
- [Digital I/O port configuration](#) (on page 8-3)
- [trigger.digout\[N\].pulsewidth](#) (on page 14-454)

trigger.digout[N].logic

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This attribute sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger <i>N</i> reset	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.digout[N].logic
trigger.digout[N].logic = logicType
```

<i>logicType</i>	The output logic of the trigger generator: - Assert a TTL-high pulse for output: <code>trigger.LOGIC_POSITIVE</code> - Assert a TTL-low pulse for output: <code>trigger.LOGIC_NEGATIVE</code>
<i>N</i>	Digital I/O trigger line: 1 to 6

Details

This attribute controls the logic that the output trigger generator uses on the given trigger line.

The output state of the digital I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Example

<pre>digio.line[4].mode = digio.MODE_TRIGGER_OUT trigger.digout[4].logic = trigger.LOGIC_NEGATIVE</pre>
Sets line 4 mode to be a trigger output and sets the output logic of the trigger event generator to negative (asserts a low pulse).

Also see

- [digio.line\[N\].mode](#) (on page 14-112)
- [digio.line\[N\].reset\(\)](#) (on page 14-114)
- [Digital I/O port configuration](#) (on page 8-3)

trigger.digout[N].pulsewidth

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This attribute describes the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger <i>N</i> reset	Configuration script	10e-6 (10 μs)

Usage

```
width = trigger.digout[N].pulsewidth
trigger.digout[N].pulsewidth = width
```

<i>width</i>	The pulse width: 0 to 100 ks
<i>N</i>	Digital I/O trigger line: 1 to 6

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely. To release the trigger line, use `trigger.digout[N].release()`.

Example

<pre>digio.line[4].mode = digio.MODE_TRIGGER_OUT trigger.digout[4].pulsewidth = 20e-6</pre>	Sets the pulse width for trigger line 4 to 20 μs.

Also see

[digio.line\[N\].mode](#) (on page 14-112)
[digio.line\[N\].reset\(\)](#) (on page 14-114)
[Digital I/O port configuration](#) (on page 8-3)
[trigger.digout\[N\].assert\(\)](#) (on page 14-452)
[trigger.digout\[N\].release\(\)](#) (on page 14-454)

trigger.digout[N].release()

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function releases an indefinite length or latched trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.digout[N].release()
```

<i>N</i>	Digital I/O trigger line: 1 to 6

Details

Releases a trigger that was asserted with an indefinite pulsewidth time. It also releases a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line is affected.

Example

<pre>digio.line[4].mode = digio.MODE_TRIGGER_OUT trigger.digout[4].release()</pre>	Releases digital I/O trigger line 4.

Also see

[digio.line\[N\].mode](#) (on page 14-112)
[Digital I/O port configuration](#) (on page 8-3)
[trigger.digout\[N\].assert\(\)](#) (on page 14-452)
[trigger.digout\[N\].pulsewidth](#) (on page 14-454)

trigger.digout[N].stimulus

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This attribute selects the event that causes a trigger to be asserted on the digital output line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Digital I/O trigger <i>N</i> reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.digout[N].stimulus
trigger.digout[N].stimulus = event
```

<i>event</i>	The event to use as a stimulus; see Details
<i>N</i>	Digital I/O trigger line: 1 to 6

Details

The digital trigger pulsewidth command determines how long the trigger is asserted.

The trigger stimulus for a digital I/O line can be set to one of the trigger events that are described in the following table.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	<code>trigger.EVENT_NONE</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
A command interface trigger (bus trigger): - Any remote interface: <code>*TRG</code> - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Scan alarm limit exceeded	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Channel closed	<code>trigger.EVENT_SCAN_CHANNEL_READY</code> (returns <code>trigger.EVENT_NOTIFY6</code>)
Scan completed	<code>trigger.EVENT_SCAN_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY8</code>)
Measure completed	<code>trigger.EVENT_SCAN_MEASURE_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY7</code>)
Limit value for scan reached	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code> (returns <code>trigger.EVENT_NOTIFY3</code>)

Example 1

```
digio.line[2].mode = digio.MODE_TRIGGER_OUT
trigger.digout[2].stimulus = trigger.EVENT_TIMER3
```

Set the stimulus for output digital trigger line 2 to be the expiration of trigger timer 3.

Example 2

```
reset()
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_RESISTANCE)
scan.create("101:109")
scan.scancount = 10
scan.bypass = scan.ON
scan.channel.stimulus = trigger.EVENT_DIGIO1
digio.line[1].mode = digio.MODE_TRIGGER_IN
trigger.digin[1].edge = trigger.EDGE_FALLING
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.digout[3].logic = trigger.LOGIC_NEGATIVE
trigger.digout[3].stimulus = trigger.EVENT_SCAN_CHANNEL_READY
trigger.model.initiate()
```

Reset the instrument.
Set channels 101 through 109 to measure 2-wire resistance.
Create a scan using channels 101 through 109.
Set the scan count to 10.
Bypass the first channel close trigger.
Set the channel close stimulus to respond to a falling edge trigger coming in on digital input line 1.
Set a digital output signal to trigger a negative pulse each time a defined scan channel is closed.
Initiate the scan.

Also see

- [digio.line\[N\].mode](#) (on page 14-112)
- [digio.line\[N\].reset\(\)](#) (on page 14-114)
- [Digital I/O port configuration](#) (on page 8-3)
- [trigger.digin\[N\].clear\(\)](#) (on page 14-448)
- [trigger.digout\[N\].assert\(\)](#) (on page 14-452)

trigger.ext.reset()

This function resets the edge, logic, and stimulus values for the EXTERNAL TRIGGER IN and EXTERNAL TRIGGER OUT lines to their default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.ext.reset()
```

Details

This function resets the following attributes to their default values:

- `trigger.extin.edge`
- `trigger.extout.logic`
- `trigger.extout.stimulus`

It also clears `trigger.extin.overflow`.

Example

```
-- Set the External Trigger In line for a rising edge
trigger.extin.edge = trigger.EDGE_RISING
-- Set the logic to negative
trigger.extout.logic = trigger.LOGIC_NEGATIVE
-- Set the stimulus to timer 3
trigger.extout.stimulus = trigger.EVENT_TIMER3
-- Print configuration (before reset)
print(trigger.extin.edge, trigger.extout.logic, trigger.extout.stimulus)
-- Reset the External Trigger In and External Trigger Out lines to default values.
trigger.ext.reset()
-- Print configuration (after reset)
print(trigger.extin.edge, trigger.extout.logic, trigger.extout.stimulus)
```

Output before reset:

```
trigger.EDGE_RISING      trigger.LOGIC_NEGATIVE  trigger.EVENT_TIMER3
```

Output after reset:

```
trigger.EDGE_FALLING     trigger.LOGIC_NEGATIVE  trigger.EVENT_NONE
```

Also see

[trigger.extin.edge](#) (on page 14-460)

[trigger.extout.logic](#) (on page 14-462)

[trigger.extout.stimulus](#) (on page 14-463)

trigger.extin.clear()

This function clears the trigger event on the EXTERNAL TRIGGER IN line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.extin.clear()
```

Details

The event detector of a trigger enters the detected state when an event is detected. This command clears the event detector, discards the history, and clears the overrun status (sets the overrun status to false).

Example

<pre>trigger.extin.clear()</pre>	Clears the trigger event detector on the EXTERNAL IN line.

Also see

[trigger.extin.overrun](#) (on page 14-460)

trigger.extin.edge

This attribute sets the type of edge that is detected as an input on the EXTERNAL TRIGGER IN trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle External I/O trigger reset	Configuration script	trigger.EDGE_FALLING

Usage

```
detectedEdge = trigger.extin.edge  
trigger.extin.edge = detectedEdge
```

detectedEdge

The trigger edge value:

- Detect falling-edge triggers as inputs: `trigger.EDGE_FALLING`
- Detect rising-edge triggers as inputs: `trigger.EDGE_RISING`
- Detect either falling or rising-edge triggers as inputs: `trigger.EDGE_EITHER`

Details

The input state of EXTERNAL TRIGGER IN is controlled by the type of edge specified by this command.

Example

```
trigger.extin.edge = trigger.EDGE_RISING
```

Sets the EXTERNAL TRIGGER IN line to detect rising-edge triggers as inputs.

Also see

[trigger.extout.logic](#) (on page 14-462)

[trigger.extout.stimulus](#) (on page 14-463)

trigger.extin.overflow

This attribute returns the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset External I/O reset	Not saved	Not applicable

Usage

```
overflow = trigger.extin.overflow
```

overflow

Trigger overrun state (`true` or `false`)

Details

If this is `true`, an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the line itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

Example

```
overrun = trigger.extin.overrun
print(overrun)
```

If there is no trigger overrun on the EXTERNAL TRIGGER IN line, the output is:
`false`

Also see

[trigger.ext.reset\(\)](#) (on page 14-458)

trigger.extin.wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.extin.wait(timeout)
```

triggered

Trigger detected: `true`
No triggers detected during the timeout period: `false`

timeout

Timeout in seconds

Details

This function pauses for up to *timeout* seconds for an input trigger. If one or more trigger events are detected since the last time `trigger.extin.wait()` or `trigger.extin.clear()` was called, this function returns a value immediately. After waiting for a trigger with this function, the event detector is automatically reset and is ready to detect the next trigger. This is true regardless of the number of events detected.

Example

```
triggered = trigger.extin.wait(3)
print(triggered)
```

Waits up to 3 s for a trigger to be detected on the EXTERNAL TRIGGER IN line, then outputs the results.
Output if no trigger is detected:
`false`
Output if a trigger is detected:
`true`

Also see

[trigger.extin.clear\(\)](#) (on page 14-459)

trigger.extout.assert()

This function asserts a trigger on the EXTERNAL TRIGGER OUT line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.extout.assert()
```

Details

Initiates a trigger event and does not wait for completion.

Example

<pre>trigger.extout.assert()</pre>	Asserts a trigger on EXTERNAL TRIGGER OUT line.

Also see

None

trigger.extout.logic

This attribute sets the output logic of the trigger event generator to positive or negative for the EXTERNAL TRIGGER OUT line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle External I/O trigger reset	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.extout.logic  
trigger.extout[N].logic = logicType
```

<pre>logicType</pre>	The output logic of the trigger generator: ■ Assert a TTL-high pulse for output: <code>trigger.LOGIC_POSITIVE</code> ■ Assert a TTL-low pulse for output: <code>trigger.LOGIC_NEGATIVE</code>

Details

This command sets the trigger event generator to assert a TTL pulse for output logic. Positive is a high pulse; negative is a low pulse.

Example

```
trigger.ext.reset()
trigger.extin.clear()
trigger.extout.logic = trigger.LOGIC_NEGATIVE
trigger.extout.stimulus = trigger.EVENT_EXTERNAL
trigger.extin.edge = trigger.EDGE_FALLING
```

Reset the EXTERNAL TRIGGER IN and EXTERNAL TRIGGER OUT line values to their defaults.
Clear any event triggers on the EXTERNAL TRIGGER IN line.
Set the output logic to negative (it asserts a low pulse).
Set the stimulus to the EXTERNAL TRIGGER IN line.
Set the external input to detect a falling edge.

Also see

[trigger.ext.reset\(\)](#) (on page 14-458)

trigger.extout.stimulus

This attribute selects the event that causes a trigger to be asserted on the EXTERNAL TRIGGER OUT line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle External I/O trigger reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.extout.stimulus
trigger.extout.stimulus = event
```

event	The event to use as a stimulus; see Details

Details

The trigger stimulus for the EXTERNAL TRIGGER OUT line can be set to one of the trigger events described in the following table.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	<code>trigger.EVENT_NONE</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
A command interface trigger (bus trigger): - Any remote interface: <code>*TRG</code> - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Scan alarm limit exceeded	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Channel closed	<code>trigger.EVENT_SCAN_CHANNEL_READY</code> (returns <code>trigger.EVENT_NOTIFY6</code>)
Scan completed	<code>trigger.EVENT_SCAN_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY8</code>)
Measure completed	<code>trigger.EVENT_SCAN_MEASURE_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY7</code>)
Limit value for scan reached	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code> (returns <code>trigger.EVENT_NOTIFY3</code>)

Example 1

```
trigger.extout.stimulus = trigger.EVENT_TIMER3
```

Set the stimulus for the EXTERNAL TRIGGER OUT line to be the expiration of trigger timer 3.

Example 2

```
reset()
channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_RESISTANCE)
scan.create("101:109")
scan.sancount = 10
scan.measure.stimulus = trigger.EVENT_EXTERNAL
trigger.extin.edge = trigger.EDGE_FALLING
trigger.extout.logic = trigger.LOGIC_NEGATIVE
trigger.extout.stimulus = trigger.EVENT_SCAN_MEASURE_COMPLETE
trigger.model.initiate()
```

Reset the instrument.
Set channels 101 through 109 to measure 2-wire resistance.
Create a scan using channels 101 through 109.
Set the scan count to 10.
Set the channel measurement stimulus to be triggered by a falling edge pulse on the EXTERNAL TRIGGER IN line.
Set the EXTERNAL TRIGGER OUT line to generate a negative pulse each time a scan channel makes a measurement.
Initiate the scan.

Also see

- [trigger.extin.edge](#) (on page 14-460)
- [trigger.extin.wait\(\)](#) (on page 14-461)
- [trigger.extout.assert\(\)](#) (on page 14-462)
- [trigger.extout.logic](#) (on page 14-462)

trigger.lanin[N].clear()

This function clears the event detector for a LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanin[N].clear()
```

N	The LAN event number: 1 to 8

Details

The trigger event detector enters the detected state when an event is detected. This function clears a trigger event detector and discards the history of the trigger packet.

This function clears all overruns associated with this LAN trigger.

Example

trigger.lanin[5].clear()	Clears the event detector with LAN event trigger 5.

Also see

- [trigger.lanin\[N\].overrun](#) (on page 14-466)

trigger.lanin[N].edge

This attribute sets the trigger operation and detection mode of the specified LAN event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EDGE_EITHER

Usage

```
edgeMode = trigger.lanin[N].edge  
trigger.lanin[N].edge = edgeMode
```

<i>edgeMode</i>	The trigger mode; see the Details for more information
<i>N</i>	The LAN event number: 1 to 8

Details

This command controls how the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values		
Mode	Trigger packets detected as input	LAN trigger packet generated for output with a...
trigger.EDGE_EITHER	Rising or falling edge (positive or negative state)	negative state
trigger.EDGE_FALLING	Falling edge (negative state)	negative state
trigger.EDGE_RISING	Rising edge (positive state)	positive state

Example

```
trigger.lanin[1].edge = trigger.EDGE_FALLING
```

 Set the edge state of LAN event 1 to falling.

Also see

[Digital I/O](#) (on page 8-1)

[TSP-Link system expansion interface](#) (on page 9-1)

trigger.lanin[N].overrun

This attribute contains the overrun status of the LAN event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	LAN trigger <i>N</i> clear	Not applicable	Not applicable

Usage

```
overrun = trigger.lanin[N].overrun
```

<i>overrun</i>	The trigger overrun state for the specified LAN packet: true or false
<i>N</i>	The LAN event number: 1 to 8

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event.

It also is not an indication of an output trigger overrun.

Example

```
overrun = trigger.lanin[5].overrun
print(overrun)
```

Checks the overrun status of a trigger on LAN5 and outputs the value, such as:

```
false
```

Also see

- [trigger.lanin\[N\].clear\(\)](#) (on page 14-465)
- [trigger.lanin\[N\].wait\(\)](#) (on page 14-467)
- [trigger.lanout\[N\].assert\(\)](#) (on page 14-468)
- [trigger.lanout\[N\].stimulus](#) (on page 14-474)

trigger.lanin[N].wait()

This function waits for an input trigger packet over LAN.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.lanin[N].wait(timeout)
```

<i>triggered</i>	Trigger detection indication: true or false
<i>N</i>	The trigger packet over LAN to wait for: 1 to 8
<i>timeout</i>	Maximum amount of time in seconds to wait for the trigger event

Details

If one or more trigger events have been detected since the last time `trigger.lanin[N].wait()` or `trigger.lanin[N].clear()` was called, this function returns immediately.

After waiting for a LAN trigger event with this function, the event detector is automatically reset and rearmed regardless of the number of events detected.

Example

```
triggered = trigger.lanin[5].wait(3)
```

Wait for a trigger event with LAN trigger 5 with a timeout of 3 s.

Also see

- [trigger.lanin\[N\].clear\(\)](#) (on page 14-465)
- [trigger.lanin\[N\].overrun](#) (on page 14-466)
- [trigger.lanout\[N\].assert\(\)](#) (on page 14-468)
- [trigger.lanout\[N\].stimulus](#) (on page 14-474)

trigger.lanout[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].assert()
```

`N`

The LAN event number: 1 to 8

Details

Generates and sends a LAN trigger packet for the LAN event number specified.

Sets the pseudo line state to the appropriate state.

The following indexes provide the listed LXI events:

- 1:LAN0
- 2:LAN1
- 3:LAN2
- ...
- 8:LAN7

Example

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

Creates a trigger with LAN trigger 5.

Also see

- [lan.lxidomain](#) (on page 14-354)
- [trigger.lanin\[N\].clear\(\)](#) (on page 14-465)
- [trigger.lanin\[N\].overrun](#) (on page 14-466)
- [trigger.lanin\[N\].wait\(\)](#) (on page 14-467)
- [trigger.lanout\[N\].assert\(\)](#) (on page 14-468)
- [trigger.lanout\[N\].ipaddress](#) (on page 14-471)
- [trigger.lanout\[N\].protocol](#) (on page 14-473)
- [trigger.lanout\[N\].stimulus](#) (on page 14-474)

trigger.lanout[N].connect()

This function prepares the event generator for outgoing trigger events.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].connect()
```

`N`

The LAN event number: 1 to 8

Details

This command prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the protocol or IP address for this event is changed.

Example

```
trigger.lanout[1].protocol = lan.PROTOCOL_MULTICAST
trigger.lanout[1].connect()
trigger.lanout[1].assert()
```

Set the protocol for LAN trigger 1 to be multicast when sending LAN triggers. Then, after connecting the LAN trigger, send a message on LAN trigger 1 by asserting it.

Also see

- [trigger.lanin\[N\].overrun](#) (on page 14-466)
- [trigger.lanin\[N\].wait\(\)](#) (on page 14-467)
- [trigger.lanout\[N\].assert\(\)](#) (on page 14-468)
- [trigger.lanout\[N\].ipaddress](#) (on page 14-471)
- [trigger.lanout\[N\].protocol](#) (on page 14-473)
- [trigger.lanout\[N\].stimulus](#) (on page 14-474)

trigger.lanout[N].connected

This attribute contains the LAN event connection state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
connected = trigger.lanout[N].connected
```

<i>connected</i>	The LAN event connection state: - Connected: true - Not connected: false
<i>N</i>	The LAN event number: 1 to 8

Details

This is set to `true` when the LAN trigger is connected and ready to send trigger events after a successful `trigger.lanout[N].connect()` command. If the LAN trigger is not ready to send trigger events, this value is `false`.

This attribute is also `false` when the `trigger.lanout[N].protocol` or `trigger.lanout[N].ipaddress` attribute is changed or when the remote connection closes the connection.

Example

```
trigger.lanout[1].protocol = lan.PROTOCOL_MULTICAST
print(trigger.lanout[1].connected)
```

Outputs `true` if connected, or `false` if not connected.

Example output:

```
false
```

Also see

[trigger.lanout\[N\].connect\(\)](#) (on page 14-469)

[trigger.lanout\[N\].ipaddress](#) (on page 14-471)

[trigger.lanout\[N\].protocol](#) (on page 14-473)

trigger.lanout[N].disconnect()

This function disconnects the LAN trigger event generator.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.lanout[N].disconnect()
```

<i>N</i>	The LAN event number: 1 to 8

Details

When this command is set for TCP connections, this closes the TCP connection.

The LAN trigger automatically disconnects when either the `trigger.lanout[N].protocol` or `trigger.lanout[N].ipaddress` attributes for this event are changed.

Also see

[trigger.lanout\[N\].ipaddress](#) (on page 14-471)

[trigger.lanout\[N\].protocol](#) (on page 14-473)

trigger.lanout[N].ipaddress

This attribute specifies the address (in dotted-decimal format) of UDP or TCP listeners.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	"0.0.0.0"

Usage

```
ipAddress = trigger.lanout[N].ipaddress
trigger.lanout[N].ipaddress = "ipAddress"
```

<i>ipAddress</i>	The LAN address for this attribute as a string in dotted decimal notation
<i>N</i>	The LAN event number: 1 to 8

Details

Sets the IP address for outgoing trigger events.

After you change this setting, you must send the connect command before outgoing messages can be sent.

Example

```
trigger.lanout[3].protocol = lan.PROTOCOL_TCP
trigger.lanout[3].ipaddress = "192.0.32.10"
trigger.lanout[3].connect()

Set the protocol for LAN trigger 3 to be TCP when sending LAN triggers.
Use IP address "192.0.32.10" to connect the LAN trigger.
```

Also see

[trigger.lanout\[N\].connect\(\)](#) (on page 14-469)

trigger.lanout[N].logic

This attribute sets the logic on which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.lanout[N].logic
trigger.lanout[N].logic = logicType
```

<i>logicType</i>	The type of logic: - Positive: trigger.LOGIC_POSITIVE - Negative: trigger.LOGIC_NEGATIVE
<i>N</i>	The LAN event number: 1 to 8

Example

```
trigger.lanout[2].logic = trigger.LOGIC_POSITIVE

Set the logic for LAN trigger line 2 to positive.
```

Also see

None

trigger.lanout[N].protocol

This attribute sets the LAN protocol to use for sending trigger messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	lan.PROTOCOL_TCP

Usage

```
protocol = trigger.lanout[N].protocol
trigger.lanout[N].protocol = protocol
```

<i>protocol</i>	The protocol to use for messages from the trigger: - lan.PROTOCOL_TCP - lan.PROTOCOL_UDP - lan.PROTOCOL_MULTICAST
<i>N</i>	The LAN event number: 1 to 8

Details

The LAN trigger listens for trigger messages on all the supported protocols. However, it uses the designated protocol for sending outgoing messages.

After you change this setting, you must re-connect the LAN trigger event generator before you can send outgoing event messages.

When multicast is selected, the trigger IP address is ignored, and event messages are sent to the multicast address 224.0.23.159.

Example

<pre>print(trigger.lanout[1].protocol)</pre>	Get LAN protocol that is being used for sending trigger messages for LAN event 1.

Also see

[trigger.lanout\[N\].connect\(\)](#) (on page 14-469)

[trigger.lanout\[N\].ipaddress](#) (on page 14-471)

trigger.lanout[N].stimulus

This attribute specifies events that cause this trigger to assert.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle	Configuration script	trigger.EVENT_NONE

Usage

```
LANevent = trigger.lanout[N].stimulus
trigger.lanout[N].stimulus = LANevent
```

LANevent	The LAN event that causes this trigger to assert; see Details for values
N	A number specifying the trigger packet over the LAN for which to set or query the trigger source: 1 to 8

Details

This attribute specifies which event causes a LAN trigger packet to be sent for this trigger. Set the event to one of the existing trigger events, which are shown in the following table.

Setting this attribute to none disables automatic trigger generation.

If any events are detected before the trigger LAN connection is sent, the event is ignored, and the action overrun is set.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	<code>trigger.EVENT_NONE</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
A command interface trigger (bus trigger): ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Scan alarm limit exceeded	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Channel closed	<code>trigger.EVENT_SCAN_CHANNEL_READY</code> (returns <code>trigger.EVENT_NOTIFY6</code>)
Scan completed	<code>trigger.EVENT_SCAN_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY8</code>)
Measure completed	<code>trigger.EVENT_SCAN_MEASURE_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY7</code>)
Limit value for scan reached	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code> (returns <code>trigger.EVENT_NOTIFY3</code>)

Example

```
trigger.lanout[5].stimulus = trigger.EVENT_TIMER1
```

Use the timer 1 trigger event as the source for LAN trigger 5 stimulus.

Also see

[trigger.lanout\[N\].connect\(\)](#) (on page 14-469)

[trigger.lanout\[N\].ipaddress](#) (on page 14-471)

trigger.model.abort()

This function stops all trigger model commands and scans on the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.abort()
```

Details

When this command is received, the instrument stops the trigger model and scans.

Example

<pre>trigger.model.abort()</pre>	Terminates all commands related to the trigger model and scans on the instrument.

Also see

[Aborting the trigger model](#) (on page 8-67)
[Effect of GPIB line events on DAQ6510](#) (on page 2-13)
[Trigger model](#) (on page 8-35)

trigger.model.getblocklist()

This function returns the settings for all trigger model blocks.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.getblocklist()
```

Details

This returns the settings for the trigger model.

If a scan is set up, this returns two trigger models that begin with START and END blocks.

Example

```
print(trigger.model.getblocklist())
```

Returns the settings for the trigger model. Example output is:

1) BUFFER_CLEAR	BUFFER: defbuffer1
2) MEASURE_DIGITIZE	BUFFER: defbuffer1 INITIAL MODE: MEAS INITIAL COUNT: 1
3) BRANCH_COUNTER	VALUE: 5 BRANCH_BLOCK: 2
4) DELAY_CONSTANT	DELAY: 1.000000000
5) BRANCH_COUNTER	VALUE: 3 BRANCH_BLOCK: 2

Also see

[trigger.model.getbranchcount\(\)](#) (on page 14-477)

trigger.model.getbranchcount()

This function returns the count value of the trigger model counter block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.getbranchcount(blockNumber)
```

<i>blockNumber</i>	The sequence of the block in the trigger model

Details

This command returns the counter value. When the counter is active, this returns the present count. If the trigger model has started or is running but has not yet reached the counter block, this value is 0.

Example

```
reset()
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 0.1)
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 10, 2)
trigger.model.initiate()
delay(1)
print(trigger.model.getbranchcount(4))
waitcomplete()
```

Reset trigger model settings.
Clear defbuffer1 at the beginning of the trigger model.
Loop and make five readings.
Delay 0.1 s.
Loop ten more times back to block 2.
Send the count command to check which count has been completed for block 4.
At end of execution, 10 readings are stored in defbuffer1.

Also see

[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_COUNTER](#) (on page 14-493)

trigger.model.initiate()

This function starts the trigger model or scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.initiate()
```

Also see

- [Trigger model](#) (on page 8-35)
- [trigger.model.abort\(\)](#) (on page 14-476)
- [trigger.model.pause\(\)](#) (on page 14-491)
- [trigger.model.resume\(\)](#) (on page 14-492)

trigger.model.load() – ConfigList

This function loads a trigger-model template configuration that uses a measure configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("ConfigList", "measureConfigList")
trigger.model.load("ConfigList", "measureConfigList", delay)
trigger.model.load("ConfigList", "measureConfigList", delay, bufferName)
```

<i>measureConfigList</i>	A string that contains the name of the measurement configuration list to use
<i>delay</i>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1.

Details

This trigger-model template incorporates a configuration list. You must set up the configuration lists before loading the trigger model. If the configuration lists change, you must resend this command.

You can also set a delay and change the reading buffer.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Example

```
reset()
dmm.measure.func = dmm.FUNC_AC_CURRENT
dmm.measure.configlist.create("MEASURE_LIST")
dmm.measure.range = 1e-3
dmm.measure.configlist.store("MEASURE_LIST")
dmm.measure.range = 10e-3
dmm.measure.configlist.store("MEASURE_LIST")
dmm.measure.range = 100e-3
dmm.measure.configlist.store("MEASURE_LIST")
trigger.model.load("ConfigList", "MEASURE_LIST")
trigger.model.initiate()
waitcomplete()
printbuffer(1, defbuffer1.n, defbuffer1.readings)
```

Reset the instrument.

Set the measure function to AC current.

Set up a configuration list named MEASURE_LIST.

Load the configuration list trigger model, using the indexes in this configuration list.

Start the trigger model.

Wait for the trigger model to complete.

Return the results from the reading buffer.

Example output:

9.9246953126e-07, 6.9921188254e-06, 3.8904102673e-05

Also see

None

trigger.model.load() – DurationLoop

This function loads a trigger-model template configuration that makes continuous measurements for a specified amount of time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("DurationLoop", duration)
trigger.model.load("DurationLoop", duration, delay)
trigger.model.load("DurationLoop", duration, delay, bufferName)
```

<i>duration</i>	The time to make measurements: 500 ns to 100 ks
<i>delay</i>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1

Details

When you load this trigger-model template, you can specify amount of time to make a measurement and the length of the delay before the measurement.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Example

```
reset()
-- Set up measure function
dmm.measure.func = dmm.FUNC_DC_CURRENT
-- Initiate readings
trigger.model.load("DurationLoop", 10, 0.01)
trigger.model.initiate()
```

Reset the instrument.
Set the instrument to measure current.
Load the duration loop trigger model to take measurements for 10 s with a 10 ms delay before each measurement.
Start the trigger model.

Also see

None

trigger.model.load() – Empty

This function clears the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.load("Empty")
```

Details

When you load this trigger-model template, any blocks that have been defined in the trigger model are cleared so the trigger model has no blocks defined.

Example

```
trigger.model.load("Empty")
print(trigger.model.getblocklist())
```

Clear the trigger model to have no blocks defined.

Output:
EMPTY

Also see

None

trigger.model.load() – GradeBinning

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function loads a trigger-model template configuration that sets up a grading operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```

trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern)
trigger.model.load("GradeBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern, bufferName)

```

<i>components</i>	The number of components to measure: 1 to 268,435,455
<i>startInLine</i>	The input line that starts the test; 5 for digital line 5, 6 for digital line 6, or 7 for external in; default is 5
<i>startDelay</i>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<i>endDelay</i>	The delay time after the measurement; default is 0 for no delay: 167 ns to 10 ks
<i>limitxHigh</i>	x is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
<i>limitxLow</i>	x is 1, 2, 3, or 4; the lower limit that the measurement is compared against
<i>limit1Pattern</i>	The bit pattern that is sent when the measurement fails limit 1: 1 to 15; default is 1
<i>limit2Pattern</i>	The bit pattern that is sent when the measurement fails limit 2: 1 to 15; default is 2
<i>limit3Pattern</i>	The bit pattern that is sent when the measurement fails limit 3: 1 to 15; default is 4
<i>limit4Pattern</i>	The bit pattern that is sent when the measurement fails limit 4: 1 to 15; default is 8

<i>allPattern</i>	The bit pattern that is sent when all limits have passed; 1 to 15; default is 15
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (<i>defbuffer1</i> or <i>defbuffer2</i>) or a user-defined buffer; defaults to <i>defbuffer1</i>

Details

This trigger-model template allows you to grade components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the pass pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Also see

None

trigger.model.load() – LogicTrigger

This function loads a trigger-model template configuration that sets up a logic trigger through the digital or external I/O.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear)
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear, delay)
trigger.model.load("LogicTrigger", digInLine, digOutLine, count, clear, delay,
    bufferName)
```

<i>digInLine</i>	The digital input line (1 to 6) or external input line (7); also, the event that the trigger model will wait on in block 1
<i>digOutLine</i>	The digital output line (1 to 6) or external input line (7)
<i>count</i>	The number of measurements the instrument will make
<i>clear</i>	To clear previously detected trigger events when entering the wait block: trigger.CLEAR_ENTER To immediately act on any previously detected triggers and not clear them (default): trigger.CLEAR_NEVER
<i>delay</i>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1

Details

This trigger model waits for a digital input or external trigger input event to occur, makes a measurement, and issues a notify event. If a digital output line is selected, a notify event asserts a digital output line. A notify event asserts the external trigger output line regardless of the line settings. You can set the line to 7 to assert only the external trigger output line, or to another setting to assert both a digital output line and the external trigger output line.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Example

```
trigger.model.load("LogicTrigger", 7, 2, 10, 0.001, defbuffer1)
```

Set up the template to use the external in line and wait for a pulse from the external in to trigger measurements. Pulse digital output line 2 and external trigger out when the measurement is complete. Make 10 measurements, with a delay of 1 ms before each measurement. Store the measurements in defbuffer1.

Also see

[trigger.digout\[N\].logic](#) (on page 14-453)

trigger.model.load() – LoopUntilEvent

This function loads a trigger-model template configuration that makes continuous measurements until the specified event occurs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear)
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear, delay)
trigger.model.load("LoopUntilEvent", triggerEvent, position, clear, delay,
    bufferName)
```

<i>triggerEvent</i>	The event that ends infinite triggering or readings set to occur before the trigger; see Details
<i>position</i>	The number of readings to make in relation to the size of the reading buffer; enter as a percentage: 0 to 100
<i>clear</i>	To clear previously detected trigger events when entering the wait block (default): trigger.CLEAR_ENTER To immediately act on any previously detected triggers and not clear them: trigger.CLEAR_NEVER
<i>delay</i>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (defbuffer1 or defbuffer2) or a user-defined buffer; defaults to defbuffer1

Details

The event constant is the event that ends infinite triggering or ends readings set to occur before the trigger and start post-trigger readings. The trigger model makes readings until it detects the event constant. After the event, it makes a finite number of readings, based on the setting of the trigger position.

The position marks the location in the reading buffer where the trigger will occur. The position is set as a percentage of the buffer capacity. The buffer captures measurements until a trigger occurs. When the trigger occurs, the buffer retains the percentage of readings specified by the position, then captures remaining readings until 100 percent of the buffer is filled. For example, if this is set to 75 for a reading buffer that holds 10,000 readings, the trigger model makes 2500 readings after it detects the source event. There are 7500 pre-trigger readings and 2500 post-trigger readings.

The instrument makes two sets of readings. The first set is made until the trigger event occurs. The second set is made after the trigger event occurs, up to the number of readings calculated by the position parameter.

You cannot have the event constant set at none when you run this trigger-model template.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

The following table lists the options that are available for *triggerEvent*.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Notify trigger block <i>N</i> (1 to 8) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
A command interface trigger (bus trigger): ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Example

```
reset()

-- Set up measure function
dmm.measure.func = dmm.FUNC_DC_CURRENT

-- Initiate readings
trigger.model.load("LoopUntilEvent", trigger.EVENT_DISPLAY, 50)
trigger.model.initiate()
```

Reset the instrument.
Set the instrument to measure current.
Load the LoopUntilEvent trigger model to make measurements until the front panel trigger key is pressed, then continue to make measurements equal to 50% of the reading buffer size.
Start the trigger model.

Also see

None

trigger.model.load() – SimpleLoop

This function loads a trigger-model template configuration that makes a specific number of measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("SimpleLoop", count)
trigger.model.load("SimpleLoop", count, delay)
trigger.model.load("SimpleLoop", count, delay, bufferName)
```

<i>count</i>	The number of measurements the instrument will make
<i>delay</i>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks
<i>bufferName</i>	Indicates the reading buffer to use; the default buffers (defbuffer1 or defbuffer2) or the name of a user-defined buffer; if no buffer is specified, defbuffer1 is used

Details

This command sets up a loop that sets a delay, makes a measurement, and then repeats the loop the number of times you define in the Count parameter.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger-model blocks in a list format.

Example

```

reset()

-- Set up measure function
dmm.measure.func = dmm.FUNC_DC_CURRENT
dmm.measure.autorange = dmm.ON
dmm.measure.nplc = 1

-- Initiate readings
trigger.model.load("SimpleLoop", 200)
trigger.model.initiate()
waitcomplete()

-- Parse index and data into three columns
print("Rdg #", "Time (s)", "Current (A)")
for i = 1, defbuffer1.n do
    print(i, defbuffer1.relativetimestamps[i], defbuffer1[i])
end

```

This example uses the SimpleLoop trigger-model template to do a capacitor test. This example produces 200 readings that have output similar to the following example:

Rdg	#	Time (s)	Current (A)
1	0	-5.6898339156e-10	
2	0.022129046	-5.6432783106e-10	
3	0.063973966	-5.6329326206e-10	
.	.	.	
198	5.133657681	-5.5518916972e-10	
199	5.155784187	-5.6363814801e-10	
200	5.177910874	-5.6070686983e-10	

Also see

None

trigger.model.load() – SortBinning**NOTE**

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function loads a trigger-model template configuration that sets up a sorting operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern)
trigger.model.load("SortBinning", components, startInLine, startDelay, endDelay,
    limit1High, limit1Low, limit1Pattern, allPattern, limit2High, limit2Low,
    limit2Pattern, limit3High, limit3Low, limit3Pattern, limit4High, limit4Low,
    limit4Pattern, bufferName)
```

<i>components</i>	The number of components to measure: 1 to 268,435,455
<i>limitxHigh</i>	<i>x</i> is limit 1, 2, 3, or 4; the upper limit that the measurement is compared against
<i>limitxLow</i>	<i>x</i> is 1, 2, 3, or 4; the lower limit that the measurement is compared against
<i>limit1Pattern</i>	The bit pattern that is sent when the measurement passes limit 1: 1 to 15; default is 1
<i>limit2Pattern</i>	The bit pattern that is sent when the measurement passes limit 2: 1 to 15; default is 2
<i>limit3Pattern</i>	The bit pattern that is sent when the measurement passes limit 3: 1 to 15; default is 4
<i>limit4Pattern</i>	The bit pattern that is sent when the measurement passes limit 4: 1 to 15; default is 8
<i>allPattern</i>	The bit pattern that is sent when all limits have failed: 1 to 15; default is 15
<i>startInLine</i>	The input line that starts the test; 5 for digital line 5, 6 for digital line 6, or 7 for external in; default is 5
<i>startDelay</i>	The delay time before each measurement; default is 0 for no delay: 167 ns to 10 ks

<i>endDelay</i>	The delay time after the measurement; default is 0 for no delay: 167 ns to 10 ks
<i>bufferName</i>	The name of the reading buffer, which may be a default buffer (<code>defbuffer1</code> or <code>defbuffer2</code>) or a user-defined buffer; defaults to <code>defbuffer1</code>

Details

This trigger-model template allows you to sort components and place them into up to four bins, based on the comparison to limits.

To set a limit as unused, set the high value for the limit to be less than the low limit.

All limit patterns and the all fail pattern are sent on digital I/O lines 1 to 4, where 1 is the least significant bit.

The rear-panel EXTERNAL TRIGGER OUT terminal is asserted at the end of each measurement.

After selecting a trigger-model template, you can view the trigger-model blocks in a graphical format by pressing the front-panel MENU key and under Trigger, selecting Configure. You can also add or delete blocks and change trigger-model settings from this screen. You can use the `trigger.model.getblocklist()` command to view the trigger model blocks in a list format.

Also see

None

trigger.model.pause()

This function pauses a running scan or trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.pause()
```

Details

This command pauses the scan or trigger model.

To continue the trigger model and the scan, send the resume command.

Example

```
reset()
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
trigger.model.pause()
delay(10)
trigger.model.resume()
waitcomplete()
print(defbuffer1.n)
```

Set up a trigger model, then run it, pause for delay of 10 seconds, then resume it.

Also see

- [trigger.model.initiate\(\)](#) (on page 14-478)
- [trigger.model.resume\(\)](#) (on page 14-492)

trigger.model.resume()

This function continues a paused scan or trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.model.resume()
```

Details

This command continues running the scan or trigger model operation if the scan or trigger model was paused.

Example

```
reset()
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
trigger.model.pause()
delay(10)
trigger.model.resume()
waitcomplete()
print(defbuffer1.n)
```

Set up a trigger model, then run it, pause for delay of 10 seconds, then resume it.

Also see

[trigger.model.initiate\(\)](#) (on page 14-478)

[trigger.model.pause\(\)](#) (on page 14-491)

trigger.model.setblock() – trigger.BLOCK_BRANCH_ALWAYS

This function defines a trigger model block that always goes to a specific block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ALWAYS, branchToBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>branchToBlock</i>	The block number to execute when the trigger model reaches the Branch Always block

Details

When the trigger model reaches a branch-always building block, it goes to the building block set by *branchToBlock*.

Example

```
trigger.model.setblock(6, trigger.BLOCK_BRANCH_ALWAYS, 20)
```

When the trigger model reaches block 6, always branch to block 20.

Also see

None

trigger.model.setblock() – trigger.BLOCK_BRANCH_COUNTER

This function defines a trigger model block that branches to a specified block a specified number of times.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_COUNTER, targetCount,  
branchToBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>targetCount</i>	The number of times to repeat
<i>branchToBlock</i>	The block number of the trigger model block to execute when the counter is less than the <i>targetCount</i> value

Details

This command defines a trigger model building block that branches to another block using a counter to iterate a specified number of times.

Counters increment every time the trigger model reaches them until they are more than or equal to the count value. At that point, the trigger model continues to the next building block in the sequence.

The counter is reset to 0 when the trigger model starts. It is incremented each time trigger model execution reaches the counter block.

If you are using remote commands, you can query the counter. The counter is incremented immediately before the branch compares the actual counter value to the set counter value. Therefore, the counter is at 0 until the first comparison. When the trigger model reaches the set counter value, branching stops and the counter value is one greater than the setting. Use `trigger.model.getbranchcount()` to query the counter.

Example

```
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 10, 2)
print(trigger.model.getbranchcount(4))
```

When the trigger model reaches this block, the trigger model returns to block 2. This repeats 10 times. An example of the return if the trigger model has reached this block 5 times is:

```
5
```

Also see

[trigger.model.getbranchcount\(\)](#) (on page 14-477)

[trigger.model.setblock\(\) — trigger.BLOCK_RESET_BRANCH_COUNT](#) (on page 14-517)

trigger.model.setblock() — trigger.BLOCK_BRANCH_DELTA

This function defines a trigger model block that goes to a specified block if the difference of two measurements meets preset criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_DELTA, targetDifference,
    branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_DELTA, targetDifference,
    branchToBlock, measureBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>targetDifference</i>	The value against which the block compares the difference between the measurements
<i>branchToBlock</i>	The block number of the trigger model block to execute when the difference between the measurements is less than or equal to the <i>targetDifference</i>
<i>measureBlock</i>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

This block calculates the difference between the last two measurements from a measure/digitize block. It subtracts the most recent measurement from the previous measurement.

The difference between the measurements is compared to the target difference. If the difference is less than the target difference, the trigger model goes to the specified branching block. If the difference is more than the target difference, the trigger model proceeds to the next block in the trigger block sequence.

If you do not define the measure/digitize block, it will compare measurements of a measure/digitize block that precedes the branch delta block. For example, if you have a measure/digitize block, a wait block, another measure/digitize block, another wait block, and then the branch delta block, the delta block compares the measurements from the second measure/digitize block. If a preceding measure/digitize block does not exist, an error occurs.

Example

```
trigger.model.setblock(5, trigger.BLOCK_BRANCH_DELTA, 0.35, 8, 3)
```

Configure trigger block 5 to branch to block 8 when the measurement difference from block 3 is less than 0.35.

Also see

[Delta block](#) (on page 8-52)

trigger.model.setblock() – trigger.BLOCK_BRANCH_LIMIT_CONSTANT

This function defines a trigger model block that goes to a specified block if a measurement meets preset criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, limitType,
    limitA, limitB, branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_CONSTANT, limitType,
    limitA, limitB, branchToBlock, measureBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>limitType</i>	The type of limit, which can be one of the following types: ■ trigger.LIMIT_ABOVE ■ trigger.LIMIT_BELOW ■ trigger.LIMIT_INSIDE ■ trigger.LIMIT_OUTSIDE
<i>limitA</i>	The lower limit that the measurement is tested against; if <i>limitType</i> is set to: ■ trigger.LIMIT_ABOVE: This value is ignored ■ trigger.LIMIT_BELOW: The measurement must be below this value ■ trigger.LIMIT_INSIDE: The low limit that the measurement is compared against ■ trigger.LIMIT_OUTSIDE: The low limit that the measurement is compared against
<i>limitB</i>	The upper limit that the measurement is tested against; if <i>limitType</i> is set to: ■ trigger.LIMIT_ABOVE: The measurement must be above this value ■ trigger.LIMIT_BELOW: This value is ignored ■ trigger.LIMIT_INSIDE: The high limit that the measurement is compared against ■ trigger.LIMIT_OUTSIDE: The high limit that the measurement is compared against
<i>branchToBlock</i>	The block number of the trigger model block to execute when the measurement meets the defined criteria
<i>measureBlock</i>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-constant-limits block goes to a branching block if a measurement meets the criteria set by this command.

The type of limit can be:

- Above: The measurement is above the value set by limit B; limit A must be set, but is ignored when this type is selected
- Below: The measurement is below the value set by limit A; limit B must be set, but is ignored when this type is selected
- Inside: The measurement is inside the values set by limits A and B; limit A must be the low value and Limit B must be the high value
- Outside: The measurement is outside the values set by limits A and B; limit A must be the low value and Limit B must be the high value

The measurement block must be a measure/digitize block that occurs in the trigger model before the branch-on-constant-limits block. The last measurement from a measure/digitize block is used.

If the limit A is more than the limit B, the values are automatically swapped so that the lesser value is used as the lower limit.

Example

```
trigger.model.setblock(5, trigger.BLOCK_BRANCH_LIMIT_CONSTANT,  
    trigger.LIMIT_ABOVE, 0.1, 1, 2)
```

Sets trigger block 5 to be a constant limit that branches to block 2 when the measurement is above the value set for limit B (which is set to 1). Note that limit A must be set but is ignored.

Also see

[Constant Limit block](#) (on page 8-48)

trigger.model.setblock() – trigger.BLOCK_BRANCH_LIMIT_DYNAMIC

This function defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, limitType,
    limitNumber, branchToBlock)
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC, limitType,
    limitNumber, branchToBlock, measureBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>limitType</i>	The type of limit, which can be one of the following types: ■ trigger.LIMIT_ABOVE ■ trigger.LIMIT_BELOW ■ trigger.LIMIT_INSIDE ■ trigger.LIMIT_OUTSIDE
<i>limitNumber</i>	The limit number (1 or 2)
<i>branchToBlock</i>	The block number of the trigger model block to execute when the measurement meets the criteria set in the configuration list
<i>measureBlock</i>	The block number of the measure/digitize block that makes the measurements to be compared; if this is 0 or undefined, the trigger model uses the previous measure/digitize block

Details

The branch-on-dynamic-limits block defines a trigger model block that goes to a specified block in the trigger model if a measurement meets user-defined criteria.

When you define this block, you set:

- The type of limit (above, below, inside, or outside the limit values)
- The limit number (you can have 1 or 2 limits)
- The block to go to if the measurement meets the criteria
- The block that makes the measurement that is compared to the limits; the last measurement from that block is used

There are two user-defined limits: limit 1 and limit 2. Both include their own high and low values, which are set using the front-panel Calculations limit settings or through commands. The results of these limit tests are recorded in the reading buffer that accompanies each stored reading.

Limit values are stored in the measure configuration list, so you can use a configuration list to step through different limit values.

The measure/digitize block must occur in the trigger model before the branch-on-dynamic-limits block. If no block is defined, the measurement from the previous measure/digitize block is used. If no previous measure/digitize block exists, an error is reported.

Example

```
trigger.model.setblock(7, trigger.BLOCK_BRANCH_LIMIT_DYNAMIC,  
    trigger.LIMIT_OUTSIDE, 2, 10, 5)
```

Configure block 7 to check if limit 2 is outside its limit values, based on the measurements made in block 5. If values are outside the measurements, branch to block 10. If the values are not outside the measurements, trigger model execution continues to block 8.

Also see

[Dynamic Limit block](#) (on page 8-50)

[dmm.measure.limit\[Y\].low.value](#) (on page 14-252)

[dmm.measure.limit\[Y\].high.value](#) (on page 14-251)

trigger.model.setblock() – trigger.BLOCK_BRANCH_ON_EVENT

This function branches to a specified block when a specified trigger event occurs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ON_EVENT, event,  
    branchToBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>event</i>	The event that must occur before the trigger model branches the specified block
<i>branchToBlock</i>	The block number of the trigger model block to execute when the specified event occurs

Details

The branch-on-event block goes to a branching block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time the trigger model reaches the branch-on-event block, it goes to the branching block.

If you set the branch event to none, an error is generated when you run the trigger model.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a Notify block before the On Event block. Set the Notify block to use the same timer as the On Event block.

The following table shows the constants for the events.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender N (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
A command interface trigger (bus trigger): ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line N (1 to 6)	<code>trigger.EVENT_DIGION</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Appropriate LXI trigger packet is received on LAN trigger object N (1 to 8)	<code>trigger.EVENT_LANN</code>
No trigger event	<code>trigger.EVENT_NONE</code>
Notify trigger block N (1 to 8) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
Notify trigger block generates a trigger event if a value in the scan is out of limits	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Trigger timer N (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
Line edge detected on TSP-Link synchronization line N (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>

Example

```
trigger.model.setblock(6, trigger.BLOCK_BRANCH_ON_EVENT, trigger.EVENT_DISPLAY, 2)
```

When the trigger model reaches this block, if the front-panel TRIGGER key has been pressed, the trigger model returns to block 2. If the TRIGGER key has not been pressed, the trigger model continues to block 7 (the next block in the trigger model).

Also see

[On event block](#) (on page 8-52)

trigger.model.setblock() – trigger.BLOCK_BRANCH_ONCE

This function causes the trigger model to branch to a specified building block the first time it is encountered in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ONCE, branchToBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>branchToBlock</i>	The block number of the trigger model block to execute when the trigger model first encounters this block

Details

The branch-once building block branches to a specified block the first time trigger model execution encounters the branch-once block. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

The once block is reset when trigger model execution reaches the idle state. Therefore, the branch-once block always executes the first time the trigger model execution encounters this block.

Example

```
trigger.model.setblock(2, trigger.BLOCK_BRANCH_ONCE, 4)
```

When the trigger model reaches block 2, the trigger model goes to block 4 instead of going in the default sequence of block 3.

Also see

[Once block](#) (on page 8-51)

trigger.model.setblock() – trigger.BLOCK_BRANCH_ONCE_EXCLUDED

This function defines a trigger model block that causes the trigger model to go to a specified building block every time the trigger model encounters it, except for the first time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BRANCH_ONCE_EXCLUDED,  
    branchToBlock)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>branchToBlock</i>	The block number of the trigger model block to execute when the trigger model encounters this block after the first encounter

Details

The branch-once-excluded block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts or is placed in idle.

Example

```
trigger.model.setblock(2, trigger.BLOCK_BRANCH_ONCE_EXCLUDED, 4)
```

When the trigger model reaches block 2 the first time, the trigger model goes to block 3. If the trigger model reaches this block again, the trigger model goes to block 4.

Also see

[Once excluded block](#) (on page 8-51)

trigger.model.setblock() – trigger.BLOCK_BUFFER_CLEAR

This function defines a trigger model block that clears the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR)  
trigger.model.setblock(blockNumber, trigger.BLOCK_BUFFER_CLEAR, bufferName)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>bufferName</i>	The name of the buffer, which must be an existing buffer; if no buffer is defined, defbuffer1 is used

Details

When trigger model execution reaches the buffer clear trigger block, the instrument empties the specified reading buffer. The specified buffer can be the default buffer or a buffer that you defined.

Example

```
trigger.model.setblock(3, trigger.BLOCK_BUFFER_CLEAR, capTest2)
```

Assign trigger block 3 to buffer clear; when the trigger model reaches block 3, it clears the reading buffer named capTest2.

Also see

[buffer.make\(\)](#) (on page 14-22)

[Buffer clear block](#) (on page 8-42)

trigger.model.setblock() – trigger.BLOCK_CONFIG_NEXT

This function recalls the settings at the next index of a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_NEXT, "configurationList")
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>configurationList</i>	A string that defines the configuration list to recall

Details

When trigger model execution reaches a configuration recall next block, the settings at the next index in the specified configuration list are restored.

The first time the trigger model encounters this block for a specific configuration list, the first index is recalled. Each subsequent time this block is encountered, the settings at the next index in the configuration list are recalled and take effect before the next step executes. When the last index in the list is reached, it returns to the first index.

The configuration list must be defined before you can use this block.

Example

```
trigger.model.setblock(5, trigger.BLOCK_CONFIG_NEXT, "measTrigList")
```

Configure trigger block 5 to load the next index in the configuration list named `measTrigList`.

Also see

[Configuration lists](#) (on page 4-99)

trigger.model.setblock() – trigger.BLOCK_CONFIG_PREV

This function defines a trigger model block that recalls the settings stored at the previous index in a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_PREV, "configurationList")
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>configurationList</i>	A string that defines the configuration list to recall

Details

The Config List Prev block defines a trigger model block that recalls the settings stored at the previous index in a configuration list.

The configuration list previous index trigger block type recalls the previous index in a configuration list. It configures the settings of the instrument based on the settings at that index. The trigger model executes the settings at that index before the next block is executed.

The first time the trigger model encounters this block, the last index in the configuration list is recalled. Each subsequent time trigger model execution reaches a configuration list previous block for this configuration list, it goes backward one index. When the first index in the list is reached, it goes to the last index in the configuration list.

The configuration list must be defined before you can use this block.

Example

```
trigger.model.setblock(8, trigger.BLOCK_CONFIG_PREV, "measTrigList")
Configure trigger block 8 to load the previous index in the configuration list named measTrigList.
```

Also see

[Configuration lists](#) (on page 4-99)

trigger.model.setblock() – trigger.BLOCK_CONFIG_RECALL

This function recalls the system settings that are stored in a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,  
    "configurationList")  
trigger.model.setblock(blockNumber, trigger.BLOCK_CONFIG_RECALL,  
    "configurationList", index)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>configurationList</i>	A string that defines the configuration list to recall
<i>index</i>	The index in the configuration list to recall; default is 1

Details

When the trigger model reaches a configuration recall block, the settings in the specified configuration list are recalled.

You can restore a specific set of configuration settings in the configuration list by defining the index.

The configuration list must be defined before you can use this block. If the configuration list changes, verify that the trigger model count is still accurate.

Example

```
trigger.model.setblock(3, trigger.BLOCK_CONFIG_RECALL, "measTrigList", 5)  
Configure trigger block 3 to load index 5 from the configuration list named measTrigList.
```

Also see

[Configuration lists](#) (on page 4-99)

trigger.model.setblock() – trigger.BLOCK_DELAY_CONSTANT

This function adds a constant delay to the execution of a trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DELAY_CONSTANT, time)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>time</i>	The amount of time to delay in seconds; 0 for no delay: 167 ns to 10 ks

Details

When trigger model execution reaches a delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made, and if any previously executed block started infinite measurements, they also continue to be made.

This delay waits for the delay time to elapse before proceeding to the next block in the trigger model.

If other delays have been set, this delay is in addition to the other delays.

Example

<pre>trigger.model.setblock(7, trigger.BLOCK_DELAY_CONSTANT, 30e-3)</pre>
Configure trigger block 7 to delay the trigger model before the next block until a delay of 30 ms elapses.

Also see

None

trigger.model.setblock() – trigger.BLOCK_DELAY_DYNAMIC

This function adds a user delay to the execution of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DELAY_DYNAMIC,
    trigger.USER_DELAY_Mn)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>n</i>	The number of the user delay, 1 to 5, set by <code>dmm.measure.userdelay[N]</code> or <code>dmm.digitize.userdelay[N]</code>

Details

When trigger model execution reaches a dynamic delay block, it stops normal measurement and trigger model operation for the time set by the delay. Background measurements continue to be made.

Each measure function can have up to five unique user delay times (M1 to M5). Digitize user delays are handled as measure user delays, so you can have a total of five measure and digitize user delays. The delay time is set by the user-delay command, which is only available over a remote interface.

Though the trigger model can be used with any function, the user delay is set per function. Make sure you are setting the delay for the function you intend to use with the trigger model. The measure user-delay settings are used with measure functions; the digitize user-delay functions are used with digitize functions.

Example

```
trigger.model.load("Empty")
dmm.measure.userdelay[1] = 5
trigger.model.setblock(1, trigger.BLOCK_DELAY_DYNAMIC, trigger.USER_DELAY_M1)
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(3, trigger.BLOCK_BRANCH_COUNTER, 10, 1)
trigger.model.initiate()
```

Set user delay 1 for measurements to 5 s.
Set trigger block 1 to a dynamic delay that calls user delay 1.
Set trigger block 2 to make or digitize a measurement.
Set trigger block 3 to branch to block 1 ten times.
Start the trigger model.

Also see

[dmm.digitize.userdelay\[N\]](#) (on page 14-190)

[dmm.measure.userdelay\[N\]](#) (on page 14-326)

trigger.model.setblock() – trigger.BLOCK_DIGITAL_IO

NOTE

This command requires a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

This function defines a trigger model block that sets the lines on the digital I/O port high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_DIGITAL_IO, bitPattern, bitMask)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>bitPattern</i>	Sets the value that specifies the output line bit pattern: 0 to 63
<i>bitMask</i>	Specifies the bit mask; if omitted, all lines are driven: 0 to 63

Details

To set the lines on the digital I/O port high or low, you can send a bit pattern. The pattern can be specified as a six-bit binary, hexadecimal, or integer value. The least significant bit maps to digital I/O line 1 and the most significant bit maps to digital I/O line 6.

The bit mask defines the bits in the pattern that are driven high or low. A binary 1 in the bit mask indicates that the corresponding I/O line should be driven according to the bit pattern. To drive all lines, specify all ones (63, 0x3F, 0b111111) or omit this parameter. If the bit for a line in the bit pattern is set to 1, the line is driven high. If the bit is set to 0 in the bit pattern, the line is driven low.

For this block to work as expected, make sure you configure the trigger type and line state of the digital line for use with the trigger model (use the digital line mode command).

Example

```
for x = 3, 6 do digio.line[x].mode = digio.MODE_DIGITAL_OUT end
trigger.model.setblock(4, trigger.BLOCK_DIGITAL_IO, 20, 60)
```

The for loop configures digital I/O lines 3 through 6 as digital outputs. Trigger block 4 is then configured with a bit pattern of 20 (digital I/O lines 3 and 5 high). The optional bit mask is specified as 60 (lines 3 through 6), so both lines 3 and 5 are driven high.

Also see

[digio.line\[N\].mode](#) (on page 14-112)

trigger.model.setblock() – trigger.BLOCK_LOG_EVENT

This function allows you to log an event in the event log when the trigger model is running.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_LOG_EVENT, eventNumber, "message")
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>eventNumber</i>	The event number: ■ trigger.LOG_INFON ■ trigger.LOG_WARNN ■ trigger.LOG_ERRORN Where <i>N</i> is 1 to 4; you can define up to four of each type You can also set trigger.LOG_WARN_ABORT, which aborts the trigger model immediately and posts a warning event log message
<i>message</i>	A string up to 31 characters

Details

This block allows you to log an event in the event log when trigger model execution reaches this block. You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

You can define the type of event (information, warning, abort model, or error). All events generated by this block are logged in the event log. Warning and error events are also displayed in a popup on the front-panel display.

Note that using this block too often in a trigger model could overflow the event log. It may also take away from the time needed to process more critical trigger model blocks.

Example

```
trigger.model.setblock(9, trigger.BLOCK_LOG_EVENT, trigger.LOG_INFO2, "Trigger model complete.")
```

Set trigger model block 9 to log an event when the trigger model completes. In the event log, the message is:
TM #1 block #9 logged: Trigger model complete.

Also see

None

trigger.model.setblock() – trigger.BLOCK_MEASURE_DIGITIZE

This function defines a trigger block that makes or digitizes a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE, bufferName)
trigger.model.setblock(blockNumber, trigger.BLOCK_MEASURE_DIGITIZE, bufferName,
    count)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>bufferName</i>	The name of the buffer, which must be an existing buffer; if no buffer is defined, defbuffer1 is used
<i>count</i>	The number of measure or digitize readings to make before moving to the next block in the trigger model; set to: ■ A specific value ■ Infinite (run continuously until stopped): <code>trigger.COUNT_INFINITE</code> ■ Stop infinite to stop the block: <code>trigger.COUNT_STOP</code> ■ Use most recent count value: <code>trigger.COUNT_AUTO</code>

Details

This block triggers measurements based on the measure function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

1. The instrument begins triggering measurements.
2. The trigger model execution waits for the measurement to be made.
3. The instrument processes the reading and places it into the specified reading buffer.

If you are defining a user-defined reading buffer, you must create it before you define this block.

When you set the count to a finite value, trigger model execution does not proceed until all operations are complete.

If you set the count to infinite, the trigger model executes subsequent blocks when the measurement is made; the triggering of measurements continues in the background until the trigger model execution reaches another measure/digitize block or until the trigger model ends. To use infinite, there must be a block after the measure/digitize block in the trigger model, such as a wait block. If there is no subsequent block, the trigger model stops, which stops measurements.

When you set the count to auto, the trigger model uses the count value that is active for the selected function instead of a specific value. You can use this with configuration lists to change the count value each time a measure/digitize block is encountered.

NOTE

If you bring in code that uses a measure or digitize block and does not define the count, the count is set to 1. For example, `trigger.model.setblock(1, trigger.BLOCK_MEASURE)` **changes to** `trigger.model.setblock(1, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1, 1)`.

Example 1

```
reset()
dmm.measure.func = dmm.FUNC_DC_VOLTAGE
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_DELAY_CONSTANT, 0)
trigger.model.setblock(3, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_INFINITE)
trigger.model.setblock(4, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
trigger.model.setblock(5, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_STOP)
trigger.model.setblock(6, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY1)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset the instrument.

Set the function to measure DC voltage.

Set block 1 to clear `defbuffer1`.

Set block 2 to set a delay of 0.

Set block 3 to make measurements infinitely.

Set block 4 to wait until the front-panel TRIGGER key is pressed.

Set block 5 to stop making measurements.

Set block 6 to send a notification.

Start the trigger model.

You must press the front-panel TRIGGER key to stop measurements.

Output the number of readings.

Example 2

```
reset()
dmm.measure.configlist.create("countactive")
dmm.measure.count = 2
dmm.measure.configlist.store("countactive") -- index1
dmm.measure.count = 10
dmm.measure.configlist.store("countactive") -- index2
dmm.measure.count = 3
dmm.measure.configlist.store("countactive") -- index3

trigger.model.setblock(1, trigger.BLOCK_CONFIG_NEXT, "countactive")
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1,
    trigger.COUNT_AUTO)
trigger.model.setblock(3, trigger.BLOCK_DELAY_CONSTANT, 1)
trigger.model.setblock(4, trigger.BLOCK_BRANCH_COUNTER, 3, 1)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset the instrument.

Set up a configuration list named `countactive`.

Set the measure count to 2 (replace `dmm.measure.count` with `dmm.digitize.count` if using a digitize function.)

Store the count in index 1.

Set the measure count to 10.

Store the count in index 2.

Set the measure count to 3.

Store the count in index 3.

Set up trigger model block 1 to call the next index from the `countactive` configuration list.

Set block 2 to measure or digitize and store the readings in `defbuffer1`, using the most recent count value.

Set block 3 to add a delay of 1 s.

Set block 4 to iterate through the trigger model 3 times, returning to block 1.

Start the trigger model.

Output the number of readings. There should be 15 readings.

Also see

[buffer.make\(\)](#) (on page 14-22)

[Measure/Digitize block](#) (on page 8-40)

trigger.model.setblock() – trigger.BLOCK_NOP

This function creates a placeholder that performs no action in the trigger model; available only using remote commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_NOP)
```

<i>blockNumber</i>	The sequence of the block in the trigger model

Details

If you remove a trigger model block, you can use this block as a placeholder for the block number so that you do not need to renumber the other blocks.

Example

<pre>trigger.model.setblock(4, trigger.BLOCK_NOP)</pre>
Set block number 4 to be a no operation block.

Also see

None

trigger.model.setblock – trigger.BLOCK_NOTIFY

This function defines a trigger model block that generates a trigger event and immediately continues to the next block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFYN)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>N</i>	The identification number of the notification: 1 to 8

Details

When trigger model execution reaches a notify block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. This assigns a stimulus somewhere else in the system. For example, you can use the notify event as the stimulus of a hardware trigger line, such as a digital I/O line.

NOTE

The TSP-Link and digital I/O options require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Example

```
digio.line[3].mode = digio.MODE_TRIGGER_OUT
trigger.model.setblock(5, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY2)
trigger.digout[3].stimulus = trigger.EVENT_NOTIFY2
```

Define trigger model block 5 to be the notify 2 event. Assign the notify 2 event to be the stimulus for digital output line 3.

Also see

Notify block

trigger.model.setblock() – trigger.BLOCK_RESET_BRANCH_COUNT

This function creates a block in the trigger model that resets a branch counter to 0.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_RESET_BRANCH_COUNT, counter)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>counter</i>	The block number of the counter that is to be reset

Details

When the trigger model reaches the Counter Reset block, it resets the count of the specified Branch on Counter block to zero.

Example

```
trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR)
trigger.model.setblock(2, trigger.BLOCK_MEASURE_DIGITIZE)
trigger.model.setblock(3, trigger.BLOCK_BRANCH_COUNTER, 5, 2)
trigger.model.setblock(4, trigger.BLOCK_DELAY_CONSTANT, 1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 3, 2)
trigger.model.setblock(6, trigger.BLOCK_RESET_BRANCH_COUNT, 3)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset trigger model settings.
Clear defbuffer1 at the beginning of the trigger model.
Loop and take 5 readings.
Delay a second.
Loop three more times back to block 2.
Reset block 3 to 0.
Start the trigger model and wait for measurements to complete.
Print the number of readings in the buffer.
Output:
15

Also see

[trigger.model.getbranchcount\(\)](#) (on page 14-477)

[trigger.model.setblock\(\) — trigger.BLOCK_BRANCH_COUNTER](#) (on page 14-493)

trigger.model.setblock() – trigger.BLOCK_WAIT

This function defines a trigger model block that waits for an event before allowing the trigger model to continue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Restore configuration Instrument reset Power cycle	Configuration script	Not applicable

Usage

```
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear, logic, event)
trigger.model.setblock(blockNumber, trigger.BLOCK_WAIT, event, clear, logic, event,
    event)
```

<i>blockNumber</i>	The sequence of the block in the trigger model
<i>event</i>	The event that must occur before the trigger block allows trigger execution to continue (see Details)
<i>clear</i>	To clear previously detected trigger events when entering the wait block: trigger.CLEAR_ENTER To immediately act on any previously detected triggers and not clear them (default): trigger.CLEAR_NEVER
<i>logic</i>	If each event must occur before the trigger model continues: trigger.WAIT_AND If at least one of the events must occur before the trigger model continues: trigger.WAIT_OR

Details

You can use the wait block to synchronize measurements with other instruments and devices.

You can set the instrument to wait for the events shown in the following table.

The event can occur before trigger model execution reaches the wait block. If the event occurs after trigger model execution starts but before the trigger model execution reaches the wait block, the trigger model records the event. By default, when trigger model execution reaches the wait block, it executes the wait block without waiting for the event to happen again (the clear parameter is set to never).

The instrument clears the memory of the recorded event when trigger model execution is at the start block and when the trigger model exits the wait block. It also clears the recorded trigger event when the clear parameter is set to enter.

All items in the list are subject to the same action; you cannot combine AND and OR logic in a single block.

You cannot leave the first event as no trigger. If the first event is not defined, the trigger model errors when you attempt to initiate it.

If you are using a timer, it must be started before it can expire. One method to start the timer in the trigger model is to include a notify block before the wait block. Set the notify block to use the same timer as the wait block.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender N (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
A command interface trigger (bus trigger): - Any remote interface: <code>*TRG</code> - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line N (1 to 6)	<code>trigger.EVENT_DIGION</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Appropriate LXI trigger packet is received on LAN trigger object N (1 to 8)	<code>trigger.EVENT_LANN</code>
No trigger event	<code>trigger.EVENT_NONE</code>
Notify trigger block N (1 to 8) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
Notify trigger block generates a trigger event if a value in the scan is out of limits	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Trigger timer N (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
Line edge detected on TSP-Link synchronization line N (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>

Example

```
trigger.model.setblock(9, trigger.BLOCK_WAIT, trigger.EVENT_DISPLAY)
```

Set trigger model block 9 to wait for a user to press the TRIGGER key on the front panel before continuing.

Also see

[Wait block](#) (on page 8-38)

trigger.model.state()

This function returns the present state of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status, status, n = trigger.model.state()
status, status, n, status, n = trigger.model.state()
```

<i>status</i>	The status of the trigger model: ■ trigger.STATE_ABORTED ■ trigger.STATE_ABORTING ■ trigger.STATE_BUILDING ■ trigger.STATE_EMPTY ■ trigger.STATE_FAILED ■ trigger.STATE_IDLE ■ trigger.STATE_PAUSED ■ trigger.STATE_RUNNING ■ trigger.STATE_WAITING
<i>n</i>	The last trigger model block that was executed

Details

This command returns the state of the trigger model. The instrument checks the state of a started trigger model every 100 ms.

This command returns the trigger state and the block that the trigger model last executed. If the trigger model supports a scan, three states and two block numbers are returned.

The trigger model states are:

- **Idle:** The trigger model is stopped
- **Running:** The trigger model is running
- **Waiting:** The trigger model has been in the same wait block for more than 100 ms
- **Empty:** The trigger model is selected, but no blocks are defined
- **Paused:** The trigger model is paused
- **Building:** Blocks have been added
- **Failed:** The trigger model is stopped because of an error
- **Aborting:** The trigger model is stopping
- **Aborted:** The trigger model is stopped

Example

```
print(trigger.model.state())
```

An example output if the trigger model is waiting and is at block 9 would be:
`trigger.STATE_WAITING trigger.STATE_EMPTY 9`

Also see

None

trigger.timer[N].clear()

This function clears the timer event detector and overrun indicator for the specified trigger timer number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.timer[N].clear()
```

N

Trigger timer number: 1 to 4

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

```
trigger.timer[1].clear()
```

Clears trigger timer 1.

Also see

[trigger.timer\[N\].count](#) (on page 14-522)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer generates a trigger event or is enabled as a timer or alarm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

<i>count</i>	Number of times to repeat the trigger: 0 to 1,048,575
<i>N</i>	Trigger timer number: 1 to 4

Details

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

Example 1

```
print(trigger.timer[1].count)
Read trigger count for timer number 1.
```

Example 2

```
reset()
trigger.timer[4].reset()
trigger.timer[4].delay = 0.5
trigger.timer[4].start.stimulus = trigger.EVENT_NOTIFY8
trigger.timer[4].start.generate = trigger.OFF
trigger.timer[4].count = 20
trigger.timer[4].enable = trigger.ON

trigger.model.load("Empty")
trigger.model.setblock(1, trigger.BLOCK_BUFFER_CLEAR, defbuffer1)
trigger.model.setblock(2, trigger.BLOCK_NOTIFY, trigger.EVENT_NOTIFY8)
trigger.model.setblock(3, trigger.BLOCK_WAIT, trigger.EVENT_TIMER4)
trigger.model.setblock(4, trigger.BLOCK_MEASURE_DIGITIZE, defbuffer1)
trigger.model.setblock(5, trigger.BLOCK_BRANCH_COUNTER, 20, 3)
trigger.model.initiate()
waitcomplete()
print(defbuffer1.n)
```

Reset the instrument.
Reset trigger timer 4.
Set trigger timer 4 to have a 0.5 s delay.
Set the stimulus for trigger timer 4 to be the notify 8 event.
Set the timer event to occur when the timer delay elapses.
Set the trigger timer 4 count to 20.
Enable trigger timer 4.

Clear the trigger model.
Set trigger model block 1 to clear the buffer.
Set trigger model block 2 to generate the notify 8 event.
Set trigger model block 3 to wait for the trigger timer 4 to occur.
Set trigger model block 4 to make or digitize a measurement and store it in default buffer 1.
Set trigger model block 5 to repeat the trigger model 20 times, starting at block 3.
Start the trigger model.
Wait until all commands are complete.
Print the number of entries in default buffer 1.

Output:
20

Also see

- [trigger.timer\[N\].clear\(\)](#) (on page 14-521)
- [trigger.timer\[N\].delay](#) (on page 14-523)
- [trigger.timer\[N\].reset\(\)](#) (on page 14-526)

trigger.timer[N].delay

This attribute sets and reads the timer delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	10e-6 (10 μs)

Usage

```
interval = trigger.timer[N].delay
trigger.timer[N].delay = interval
```

<i>interval</i>	Delay interval in seconds: 8 μs to 100 ks
<i>N</i>	Trigger timer number: 1 to 4

Details

Once the timer is enabled, each time the timer is triggered, it uses this delay period.

Assigning a value to this attribute is equivalent to:

```
trigger.timer[N].delaylist = {interval}
```

This creates a delay list of one value.

Reading this attribute returns the delay interval that is used the next time the timer is triggered.

If you use the trigger timer with a trigger model, make sure the trigger timer delay is set so that the readings are paced correctly.

Example

```
trigger.timer[1].delay = 50e-6
```

Set the trigger timer 1 to delay for 50 μ s.

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 14-526)

trigger.timer[N].delaylist

This attribute sets an array of timer intervals.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	10e-6 (10 μ s)

Usage

```
intervals = trigger.timer[N].delaylist  
trigger.timer[N].delaylist = intervals
```

<i>intervals</i>	Table of delay intervals in seconds
<i>N</i>	Trigger timer number: 1 to 4

Details

Each time the timer is triggered after it is enabled, it uses the next delay period from the array. The default value is an array with one value of 10 μ s.

After all elements in the array have been used, the delays restart at the beginning of the list.

If the array contains more than one element, the average of the delay intervals in the list must be ≥ 50 μ s.

Example

```
trigger.timer[3].delaylist = {50e-6, 100e-6, 150e-6}
DelayList = trigger.timer[3].delaylist
for x = 1, table.getn(DelayList) do
    print(DelayList[x])
end
```

Set a delay list on trigger timer 3 with three delays (50 μ s, 100 μ s, and 150 μ s).
Read the delay list on trigger timer 3.
Output:
5e-05
0.0001
0.00015

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 14-526)

trigger.timer[N].enable

This attribute enables the trigger timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	trigger.OFF

Usage

```
state = trigger.timer[N].enable
trigger.timer[N].enable = state
```

state	Disable the trigger timer: trigger.OFF Enable the trigger timer: trigger.ON
N	Trigger timer number: 1 to 4

Details

When this command is set to on, the timer performs the delay operation.

When this command is set to off, there is no timer on the delay operation.

You must enable a timer before it can use the delay settings or the alarm configuration. For expected results from the timer, it is best to disable the timer before changing a timer setting, such as delay or start seconds.

To use the timer as a simple delay or pulse generator with digital I/O lines, make sure the timer start time in seconds and fractional seconds is configured for a time in the past. To use the timer as an alarm, configure the timer start time in seconds and fractional seconds for the desired alarm time.

Example

```
trigger.timer[3].enable = trigger.ON
```

Enable the trigger timer for timer 3.

Also see

None

trigger.timer[N].reset()

This function resets trigger timer settings to their default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.timer[N].reset()
```

<i>N</i>	Trigger timer number: 1 to 4

Details

The `trigger.timer[N].reset()` function resets the following attributes to their default values:

- `trigger.timer[N].count`
- `trigger.timer[N].delay`
- `trigger.timer[N].delaylist`
- `trigger.timer[N].enable`
- `trigger.timer[N].start.fractionalseconds`
- `trigger.timer[N].start.generate`
- `trigger.timer[N].start.seconds`
- `trigger.timer[N].stimulus`

It also clears `trigger.timer[N].overrun`.

Example

<code>trigger.timer[1].reset()</code>	Resets the attributes associated with timer 1 to their default values.

Also see

[trigger.timer\[N\].count](#) (on page 14-522)
[trigger.timer\[N\].delay](#) (on page 14-523)
[trigger.timer\[N\].delaylist](#) (on page 14-524)
[trigger.timer\[N\].enable](#) (on page 14-525)
[trigger.timer\[N\].start.fractionalseconds](#) (on page 14-527)
[trigger.timer\[N\].start.generate](#) (on page 14-528)
[trigger.timer\[N\].start.overrun](#) (on page 14-529)
[trigger.timer\[N\].start.seconds](#) (on page 14-530)
[trigger.timer\[N\].start.stimulus](#) (on page 14-530)

trigger.timer[N].start.fractionalseconds

This attribute configures the fractional seconds of an alarm or a time in the future when the timer will start.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	0

Usage

```
time = trigger.timer[N].start.fractionalseconds
trigger.timer[N].start.fractionalseconds = time
```

<i>time</i>	The time in fractional seconds: 0 to <1 s
<i>N</i>	Trigger timer number: 1 to 4

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time that has passed.

Example

<code>trigger.timer[1].start.fractionalseconds = 0.4</code>	Set the trigger timer to start in 0.4 s.

Also see

[trigger.timer\[N\].start.generate](#) (on page 14-528)

trigger.timer[N].start.generate

This attribute specifies when timer events are generated.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	trigger.OFF

Usage

```
state = trigger.timer[N].start.generate
trigger.timer[N].start.generate = state
```

state	Generate a timer event when the timer delay elapses: trigger.OFF Generate a timer event when the timer starts and when the delay elapses: trigger.ON
N	Trigger timer number: 1 to 4

Details

When this is set to on, a trigger event is generated immediately when the timer is triggered.

When it is set to off, a trigger event is generated when the timer elapses. This generates the event trigger.EVENT_TIMERN.

Example

```
trigger.timer[4].reset()
trigger.timer[4].delay = 0.5
trigger.timer[4].start.stimulus = trigger.EVENT_NOTIFY8
trigger.timer[4].start.generate = trigger.OFF
trigger.timer[4].count = 20
trigger.timer[4].enable = trigger.ON
```

Reset trigger timer 4.
Set trigger timer 4 to have a 0.5 s delay.
Set the stimulus for trigger timer 4 to be the notify 8 event.
Set the timer event to occur when the timer delay elapses.
Set the trigger timer 4 count to 20.
Enable trigger timer 4.

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 14-526)

trigger.timer[N].start.overrun

This attribute indicates if an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Trigger timer <i>N</i> reset	Not applicable	Not applicable

Usage

```
state = trigger.timer[N].start.overrun
```

<i>state</i>	The trigger overrun state: <code>true</code> or <code>false</code>
<i>N</i>	Trigger timer number: 1 to 4

Details

This command indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Example

<pre>print(trigger.timer[1].start.overrun)</pre>	If an event was ignored, the output is <code>true</code> . If the event was not ignored, the output is <code>false</code> .

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 14-526)

trigger.timer[N].start.seconds

This attribute configures the seconds of an alarm or a time in the future when the timer will start.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	0 (0 s)

Usage

```
time = trigger.timer[N].start.seconds  
trigger.timer[N].start.seconds = time
```

<i>time</i>	The time in seconds: 0 to 2,147,483,647
<i>N</i>	Trigger timer number: 1 to 4

Details

This command configures the alarm of the timer.

When the timer is enabled, the timer starts immediately if the timer is configured for a start time that has passed.

Example

```
trigger.timer[1].start.seconds = localnode.gettime() + 30  
trigger.timer[1].enable = trigger.ON
```

Set the trigger timer to start 30 s from the time when the timer is enabled.

Also see

None

trigger.timer[N].start.stimulus

This attribute describes the event that starts the trigger timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Trigger timer <i>N</i> reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.timer[N].start.stimulus  
trigger.timer[N].start.stimulus = event
```

<i>event</i>	The event that starts the trigger timer; see Details
<i>N</i>	Trigger timer number: 1 to 4

Details

Set the stimulus to any trigger event to start the timer when that event occurs.

Set the stimulus to none to disable event processing and use the timer as a timer or alarm based on the start time.

Trigger events are described in the table below.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	<code>trigger.EVENT_NONE</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
A command interface trigger (bus trigger): - Any remote interface: <code>*TRG</code> - GPIB only: GET bus command - USB only: A USBTMC TRIGGER message - VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Scan alarm limit exceeded	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Channel closed	<code>trigger.EVENT_SCAN_CHANNEL_READY</code> (returns <code>trigger.EVENT_NOTIFY6</code>)
Scan completed	<code>trigger.EVENT_SCAN_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY8</code>)
Measure completed	<code>trigger.EVENT_SCAN_MEASURE_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY7</code>)
Limit value for scan reached	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code> (returns <code>trigger.EVENT_NOTIFY3</code>)

Example

```

digio.line[3].mode = digio.MODE_TRIGGER_IN
trigger.timer[1].delay = 3e-3
trigger.timer[1].start.stimulus = trigger.EVENT_DIGIO3

```

Set digital I/O line 3 to be a trigger input.
Set timer 1 to delay for 3 ms.
Set timer 1 to start the timer when an event is detected on digital I/O line 3.

Also see

None

trigger.timer[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.timer[N].wait(timeout)
```

<i>triggered</i>	Trigger detection indication
<i>N</i>	Trigger timer number: 1 to 4
<i>timeout</i>	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = trigger.timer[3].wait(10)
print(triggered)
```

Waits up to 10 s for a trigger on timer 3.
If `false` is returned, no trigger was detected during the 10 s timeout.
If `true` is returned, a trigger was detected.

Also see

[trigger.timer\[N\].clear\(\)](#) (on page 14-521)

trigger.tsplinkin[N].clear()

This function clears the event detector for a LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.tsplinkin[N].clear()
```

<i>N</i>	The trigger line: 1 to 3 to clear

Details

The trigger event detector enters the detected state when an event is detected. When this command is sent, the instrument:

- Clears the trigger event detector
- Discards the history of the trigger line
- Clears the `trigger.tsplinkin[N].overrun` attribute

Example

```
tsplink.line[2].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkin[2].clear()
```

Clears the trigger event on TSP-Link line 2.

Also see

[trigger.tsplinkin\[N\].overrun](#) (on page 14-536)
[tsplink.line\[N\].mode](#) (on page 14-545)

trigger.tsplinkin[N].edge

This attribute indicates which trigger edge controls the trigger event detector for a trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	trigger.EDGE_FALLING

Usage

```
detectedEdge = trigger.tsplinkin[N].edge
trigger.tsplinkin[N].edge = detectedEdge
```

<i>detectedEdge</i>	The trigger mode: ▪ Detect falling-edge triggers as inputs: <code>trigger.EDGE_FALLING</code> ▪ Detect rising-edge triggers as inputs: <code>trigger.EDGE_RISING</code> ▪ Detect either falling or rising-edge triggers as inputs: <code>trigger.EDGE_EITHER</code>
<i>N</i>	The trigger line: 1 to 3

Details

When the edge is detected, the instrument asserts a TTL-low pulse for the output.

The output state of the I/O line is controlled by the trigger logic. The user-specified output state of the line is ignored.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkin[3].edge = trigger.EDGE_RISING
```

Sets synchronization line 3 to detect rising edge triggers as input.

Also see

- [digio.writeport\(\)](#) (on page 14-117)
- [tsplink.line\[N\].mode](#) (on page 14-545)
- [tsplink.line\[N\].reset\(\)](#) (on page 14-545)

trigger.tsplinkin[N].overrun

This attribute indicates if the event detector ignored an event while in the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup TSP-Link line <i>N</i> clear	Not applicable	Not applicable

Usage

```
overrun = trigger.tsplinkin[N].overrun
```

<i>overrun</i>	Trigger overrun state
<i>N</i>	The trigger line: 1 to 3

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself.

It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event. It also is not an indication of an output trigger overrun.

Example

```
print(trigger.tsplinkin[1].overrun)
```

If an event on line 1 was ignored, displays `true`; if no additional event occurred, displays `false`.

Also see

None

trigger.tsplinkin[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.tsplinkin[N].wait(timeout)
```

<i>triggered</i>	Trigger detection indication; set to one of the following values: - A trigger is detected during the timeout period: <code>true</code> - A trigger is not detected during the timeout period: <code>false</code>
<i>N</i>	The trigger line: 1 to 3
<i>timeout</i>	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events are detected since the last time this command or `trigger.tsplinkin[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
triggered = trigger.tsplinkin[3].wait(10)
print(triggered)
```

Waits up to 10 s for a trigger on TSP-Link line 3.
If `false` is returned, no trigger was detected during the 10-s timeout.
If `true` is returned, a trigger was detected.

Also see

[trigger.tsplinkin\[N\].clear\(\)](#) (on page 14-534)
[tsplink.line\[N\].mode](#) (on page 14-545)

trigger.tsplinkout[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.tsplinkout[N].assert()
```

<i>N</i>	The trigger line: 1 to 3

Details

Initiates a trigger event and does not wait for completion. The set pulse width determines how long the trigger is asserted.

Example

```
tsplink.line[2].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[2].assert()
```

Asserts trigger on trigger line 2.

Also see

[tsplink.line\[N\].mode](#) (on page 14-545)

trigger.tsplinkout[N].logic

This attribute defines the trigger output with output logic for a trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	trigger.LOGIC_NEGATIVE

Usage

```
logicType = trigger.tsplinkout[N].logic
trigger.tsplinkout[N].logic = logicType
```

<i>logicType</i>	The output logic of the trigger generator: - Assert a TTL-high pulse for output: <code>trigger.LOGIC_POSITIVE</code> - Assert a TTL-low pulse for output: <code>trigger.LOGIC_NEGATIVE</code>
<i>N</i>	The trigger line: 1 to 3

Details

This attribute controls the logic that the output trigger generator uses on the given trigger line.

The output state of the digital I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[3].logic = trigger.LOGIC_POSITIVE
```

Sets the trigger logic for synchronization line 3 to output a positive pulse.

Also see

[trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-537)

[tsplink.line\[N\].mode](#) (on page 14-545)

trigger.tsplinkout[N].pulsewidth

This attribute sets the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	10e-6 (10 μ s)

Usage

```
width = trigger.tsplinkout[N].pulsewidth
trigger.tsplinkout[N].pulsewidth = width
```

<i>width</i>	The pulse width in s: 0.0 to 100,000
<i>N</i>	The trigger line: 1 to 3

Details

Setting the pulse width to 0 asserts the trigger indefinitely.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[3].pulsewidth = 20e-6
```

Sets pulse width for trigger line 3 to 20 μ s.

Also see

- [trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-537)
- [trigger.tsplinkout\[N\].release\(\)](#) (on page 14-539)
- [tsplink.line\[N\].mode](#) (on page 14-545)

trigger.tsplinkout[N].release()

This function releases a latched trigger on the given TSP-Link trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.tsplinkout[N].release()
```

<i>N</i>	The trigger line: 1 to 3

Details

Releases a trigger that was asserted with an indefinite pulse width. It also releases a trigger that was latched in response to receiving a synchronous mode trigger.

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN
trigger.tsplinkout[3].release()
```

Releases trigger line 3.

Also see

- [trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-537)
- [tsplink.line\[N\].mode](#) (on page 14-545)

trigger.tsplinkout[N].stimulus

This attribute specifies the event that causes the synchronization line to assert a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	trigger.EVENT_NONE

Usage

```
event = trigger.tsplinkout[N].stimulus
trigger.tsplinkout[N].stimulus = event
```

event	The event identifier for the triggering event (see Details)
<i>N</i>	The trigger line: 1 to 3

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to `trigger.EVENT_NONE`.

Do not use this attribute when triggering under script control. Use `trigger.tsplinkout[N].assert()` instead.

The *event* parameters that you can use are described in the table below.

NOTE

The options in the following table for digital I/O, GPIB, and TSP-Link require a communications accessory card to be installed in the instrument. Accessory cards include the KTTI-GPIB, KTTI-TSP, and KTTI-RS232.

Trigger events	
Event description	Event constant
No trigger event (make measurement immediately)	<code>trigger.EVENT_NONE</code>
Front-panel TRIGGER key press	<code>trigger.EVENT_DISPLAY</code>
Notify trigger block <i>N</i> (1 to 3) generates a trigger event when the trigger model executes it	<code>trigger.EVENT_NOTIFYN</code>
A command interface trigger (bus trigger): ▪ Any remote interface: *TRG ▪ GPIB only: GET bus command ▪ USB only: A USBTMC TRIGGER message ▪ VXI-11: VXI-11 command <code>device_trigger</code>	<code>trigger.EVENT_COMMAND</code>
Line edge (either rising, falling, or either based on the configuration of the line) detected on digital input line <i>N</i> (1 to 6)	<code>trigger.EVENT_DIGION</code>
Line edge detected on TSP-Link synchronization line <i>N</i> (1 to 3)	<code>trigger.EVENT_TSPLINKN</code>
Appropriate LXI trigger packet is received on LAN trigger object <i>N</i> (1 to 8)	<code>trigger.EVENT_LANN</code>
Analog trigger	<code>trigger.EVENT_ANALOGTRIGGER</code>
Trigger event blender <i>N</i> (1 to 2), which combines trigger events	<code>trigger.EVENT_BLENDERN</code>
Trigger timer <i>N</i> (1 to 4) expired	<code>trigger.EVENT_TIMERN</code>
External in trigger	<code>trigger.EVENT_EXTERNAL</code>
Scan alarm limit exceeded	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code>
Channel closed	<code>trigger.EVENT_SCAN_CHANNEL_READY</code> (returns <code>trigger.EVENT_NOTIFY6</code>)
Scan completed	<code>trigger.EVENT_SCAN_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY8</code>)
Measure completed	<code>trigger.EVENT_SCAN_MEASURE_COMPLETE</code> (returns <code>trigger.EVENT_NOTIFY7</code>)
Limit value for scan reached	<code>trigger.EVENT_SCAN_ALARM_LIMIT</code> (returns <code>trigger.EVENT_NOTIFY3</code>)

Example

```
print(trigger.tsplinkout[3].stimulus)
```

Outputs the event that will start action on TSP-Link trigger line 3.

Also see

[trigger.tsplinkout\[N\].assert\(\)](#) (on page 14-537)

[tsplink.line\[N\].reset\(\)](#) (on page 14-545)

trigger.wait()

This function waits for a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
triggered = trigger.wait(timeout)
```

<i>triggered</i>	A trigger was detected during the timeout period: <code>true</code> No triggers were detected during the timeout period: <code>false</code>
<i>timeout</i>	Maximum amount of time in seconds to wait for the trigger

Details

This function waits up to *timeout* seconds for a trigger on the active command interface. A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 device_trigger method is invoked (VXI-11 only)
- A USBTMC trigger message is received (USB only)
- A *TRG message is received

If one or more of these trigger events were previously detected, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

<pre>triggered = trigger.wait(10) print(triggered)</pre>	Waits up to 10 s for a trigger. If <code>false</code> is returned, no trigger was detected during the 10 s timeout. If <code>true</code> is returned, a trigger was detected.

Also see

[trigger.clear\(\)](#) (on page 14-447)

tsplink.group

This attribute contains the group number of a TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
groupNumber = tsplink.group
tsplink.group = groupNumber
```

<i>groupNumber</i>	The group number of the TSP-Link node: 0 to 64

Details

To remove the node from all groups, set the attribute value to 0.

When the node is turned off, the group number for that node changes to 0.

The master node can be assigned to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

Example

```
tsplink.group = 3
Assign the instrument to TSP-Link group number 3.
```

Also see

[Using groups to manage nodes on a TSP-Link system](#) (on page 9-10)

tsplink.initialize()

This function initializes all instruments and enclosures in the TSP-Link system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
nodesFound = tsplink.initialize()
tsplink.initialize()
tsplink.initialize(expectedNodes)
```

<i>nodesFound</i>	The number of nodes found on the system, including the node on which the command is running
<i>expectedNodes</i>	The number of nodes expected on the system: 1 to 32

Details

This function regenerates the system configuration information regarding the nodes connected to the TSP-Link system. You must initialize the system after making configuration changes. You need to initialize the system after you:

- Turn off power or reboot any instrument in the system
- Change node numbers on any instrument in the system
- Rearrange or disconnect the TSP-Link cable connections between instruments

If the only node on the TSP-Link network is the one running the command and *expectedNodes* is not provided, this function generates an error event. If you set *expectedNodes* to 1, the node is initialized.

If you include *expectedNodes*, if *nodesFound* is less than *expectedNodes*, an error event is generated.

NOTE

If any TSP-Link cabled node is powered off, initialize will fail.

Example

```
nodesFound = tsplink.initialize(2)
print("Nodes found = " .. nodesFound)
```

Perform a TSP-Link initialization and indicate how many nodes are found.

Example output if two nodes are found:

```
Nodes found = 2
```

Example output if fewer nodes are found and if `localnode.showevents = 7`:

```
1219, TSP-Link found fewer nodes than expected
```

```
Nodes found = 1
```

Also see

[Initializing the TSP-Link system](#) (on page 9-6)

[localnode.showevents](#) (on page 14-362)

[tsplink.node](#) (on page 14-548)

[tsplink.state](#) (on page 14-550)

tsplink.line[N].mode

This attribute defines the trigger operation of a TSP-Link line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle TSP-Link line <i>N</i> reset	Configuration script	tsplink.MODE_DIGITAL_OPEN_DRAIN

Usage

```
mode = tsplink.line[N].mode  
tsplink.line[N].mode = mode
```

<i>mode</i>	The trigger mode; see Details
<i>N</i>	The trigger line: 1 to 3

Details

This command defines whether or not the line is used as a digital or trigger control line and if it is an input or output.

The line mode can be set to the following options:

- TSP-Link digital open drain line: `tsplink.MODE_DIGITAL_OPEN_DRAIN`
- TSP-Link trigger open drain line: `tsplink.MODE_TRIGGER_OPEN_DRAIN`
- TSP-Link trigger synchronous master: `tsplink.MODE_SYNCHRONOUS_MASTER`
- TSP-Link trigger synchronous acceptor: `tsplink.MODE_SYNCHRONOUS_ACCEPTOR`

Example

```
tsplink.line[3].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN  
Sets the trigger mode for synchronization line 3 as a trigger open drain line.
```

Also see

[trigger.tsplinkin\[N\].edge](#) (on page 14-535)

[trigger.tsplinkout\[N\].logic](#) (on page 14-538)

tsplink.line[N].reset()

This function resets some of the TSP-Link trigger attributes to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.line[N].reset()
```

<i>N</i>	The trigger line: 1 to 3

Details

The `tsplink.line[N].reset()` function resets the following attributes to their default values:

- `tsplink.line[N].mode`
- `trigger.tsplinkin[N].edge`
- `trigger.tsplinkout[N].logic`
- `trigger.tsplinkout[N].pulsewidth`
- `trigger.tsplinkout[N].stimulus`

This also clears `trigger.tsplinkin[N].overrun`.

Example

```
tsplink.line[3].reset()
```

Resets TSP-Link trigger line 3 attributes to default values.

Also see

[trigger.tsplinkin\[N\].edge](#) (on page 14-535)
[trigger.tsplinkin\[N\].overrun](#) (on page 14-536)
[trigger.tsplinkout\[N\].logic](#) (on page 14-538)
[trigger.tsplinkout\[N\].pulsewidth](#) (on page 14-538)
[trigger.tsplinkout\[N\].stimulus](#) (on page 14-540)
[tsplink.line\[N\].mode](#) (on page 14-545)

tsplink.line[N].state

This attribute reads or writes the digital state of a TSP-Link synchronization line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	tsplink.STATE_HIGH

Usage

```
lineState = tsplink.line[N].state
tsplink.line[N].state = lineState
```

<i>lineState</i>	The state of the synchronization line: ■ Low: <code>tsplink.STATE_LOW</code> ■ High: <code>tsplink.STATE_HIGH</code>
<i>N</i>	The trigger line: 1 to 3

Details

Use `tsplink.writeport()` to write to all TSP-Link synchronization lines.

The reset function does not affect the present states of the TSP-Link trigger lines.

Example

```
lineState = tsplink.line[3].state
print(lineState)
```

Assume line 3 is set high, and then the state is read.
Output:
`tsplink.STATE_HIGH`

Also see

[tsplink.line\[N\].mode](#) (on page 14-545)
[tsplink.writeport\(\)](#) (on page 14-551)

tsplink.master

This attribute reads the node number assigned to the master node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
masterNodeNumber = tsplink.master
```

```
masterNodeNumber
```

 The node number of the master node: 1 to 63

Details

This attribute returns the node number of the master in a set of instruments connected using TSP-Link.

Example

```
LinkMaster = tsplink.master
```

Store the TSP-Link master node number in a variable called `LinkMaster`.

Also see

[tsplink.initialize\(\)](#) (on page 14-543)

tsplink.node

This attribute defines the node number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	2

Usage

```
nodeNumber = tsplink.node
tsplink.node = nodeNumber
```

<i>nodeNumber</i>	The node number of the instrument or enclosure: 1 to 63

Details

This command sets the TSP-Link node number and saves the value in nonvolatile memory.

Changes to the node number do not take effect until `tsplink.reset()` from an earlier TSP-Link instrument or `tsplink.initialize()` is executed on any node in the system.

Each node connected to the TSP-Link system must be assigned a different node number.

Example

<pre>tsplink.node = 3</pre>
Sets the TSP-Link node for this instrument to number 3.

Also see

- [tsplink.initialize\(\)](#) (on page 14-543)
- [tsplink.state](#) (on page 14-550)

tsplink.readport()

This function reads the TSP-Link trigger lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readport()
```

data	Numeric value that indicates which lines are set

Details

The binary equivalent of the returned value indicates the input pattern on the I/O port. The least significant bit of the binary number corresponds to line 1 and the value of bit 3 corresponds to line 3. For example, a returned value of 2 has a binary equivalent of 010. This indicates that line 2 is high (1), and that the other two lines are low (0).

Example

```
data = tsplink.readport()
print(data)
```

Reads state of all three TSP-Link lines.
Assuming line 2 is set high, the output is:
2.000000e+00
(binary 010)
The format of the output may vary depending on the ASCII precision setting.

Also see

- [Triggering using TSP-Link trigger lines](#) (on page 9-8)
- [tsplink.line\[N\].state](#) (on page 14-546)
- [tsplink.writeport\(\)](#) (on page 14-551)

tsplink.state

This attribute describes the TSP-Link online state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
state = tsplink.state
```

state	TSP-Link state: online or offline

Details

When the instrument power is first turned on, the state is `offline`. After `tsplink.initialize()` or `tsplink.reset()` is successful, the state is `online`.

Example

```
state = tsplink.state
print(state)
```

Read the state of the TSP-Link system. If it is online, the output is:
online

Also see

[tsplink.node](#) (on page 14-548)

tsplink.writeport()

This function writes to all TSP-Link synchronization lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writeport(data)
```

data	Value to write to the port: 0 to 7

Details

The binary representation of `data` indicates the output pattern that is written to the I/O port. For example, a data value of 2 has a binary equivalent of 010. Line 2 is set high (1), and the other two lines are set low (0).

The `reset()` function does not affect the present states of the trigger lines.

Example

```
tsplink.writeport(3)
```

Sets the synchronization lines 1 and 2 high (binary 011).

Also see

[tsplink.line\[N\].state](#) (on page 14-546)

tspnet.clear()

This function clears any pending output data from the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.clear(connectionID)
```

connectionID	The connection ID returned from <code>tspnet.connect()</code>

Details

This function clears any pending output data from the device. No data is returned to the caller and no data is processed.

Example

```
tspnet.write(testdevice, "print([[hello]])")
print(tspnet.readavailable(testdevice))
tspnet.clear(testdevice)
print(tspnet.readavailable(testdevice))
```

Write data to a device, then print how much is available.
Output:
6.00000e+00

Clear data and print how much data is available again.
Output:
0.00000e+00

Also see

- [tspnet.connect\(\)](#) (on page 14-553)
- [tspnet.readavailable\(\)](#) (on page 14-559)
- [tspnet.write\(\)](#) (on page 14-567)

tspnet.connect()

This function establishes a network connection with another LAN instrument or device through the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
connectionID = tspnet.connect("ipAddress")
connectionID = tspnet.connect("ipAddress", portNumber, "initString")
connectionID = tspnet.connect("ipAddress", portNumber, "initString", useTLS)
```

connectionID	The connection ID to be used as a handle in all other <code>tspnet</code> function calls
ipAddress	IP address to which to connect in a string; accepts IP address or host name when trying to connect
portNumber	Port number (default 5025)
initString	Initialization string to send to <code>ipAddress</code> ; when <code>useTLS</code> is set to 1; set this to null
useTLS	Determine if the TLS security protocol is negotiated when connecting the instrument to the host or IP address: - Do not negotiate the security protocol (default): 0 - Negotiate the security protocol (see Details): 1

Details

This command connects a device to another device through the LAN interface. If the `portNumber` is 23, the interface uses the Telnet protocol and sets appropriate termination characters to communicate with the device.

If a `portNumber` and `initString` are provided, it is assumed that the remote device is not TSP-enabled. The DAQ6510 does not perform any extra processing, prompt handling, error handling, or sending of commands. In addition, the `tspnet.tsp.*` commands cannot be used on devices that are not TSP-enabled.

If neither a `portNumber` nor an `initString` is provided, the remote device is assumed to be a Keithley TSP-enabled device. Depending on the state of the `tspnet.tsp.abortonconnect` attribute, the DAQ6510 sends an `abort` command to the remote device on connection.

The DAQ6510 also enables TSP prompts on the remote device and event management. The DAQ6510 places remote errors and events from the TSP-enabled device in its own event queue and prefaces these events with `Remote Error`, followed by an event description.

Do not manually change either the prompt functionality (`localnode.prompts`) or show events by changing `localnode.showerrors` or `localnode.showevents` on the remote TSP-enabled device. If you do this, subsequent `tspnet.tsp.*` commands using the connection may fail.

When you are connecting to a host or IP address, you can add a layer of transport layer security (TLS). When security is enabled, the instrument negotiates the security protocol when connecting to the host or IP address. This security protocol is used when using `tspnet.write()` to send data or `tspnet.read()` to receive data. Typically, the port is 443 if TLS is enabled.

You can simultaneously connect to a maximum of 32 remote devices.

Example 1

```
instrumentID = tspnet.connect("192.0.2.1")
if instrumentID then
  -- Use instrumentID as needed here
  tspnet.disconnect(instrumentID)
end
```

Connect to a TSP-enabled device.

Example 2

```
instrumentID = tspnet.connect("192.0.2.1", 1394, "*rst\r\n")
if instrumentID then
  -- Use instrumentID as needed here
  tspnet.disconnect(instrumentID)
end
```

Connect to a device that is not TSP-enabled.

Example 3

```
instrumentID = tspnet.connect("192.0.2.1", 443, "", 1)
if instrumentID then
  -- Use instrumentID as needed here
  tspnet.disconnect(instrumentID)
end
```

Connect to a TSP-enabled device with TLS enabled using port 443.

Also see

[localnode.prompts](#) (on page 14-358)
[localnode.showevents](#) (on page 14-362)
[tspnet.tsp.abortonconnect](#) (on page 14-564)
[tspnet.disconnect\(\)](#) (on page 14-555)

tspnet.disconnect()

This function disconnects a specified TSP-Net session.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

tspnet.disconnect(*connectionID*)

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

Details

This function disconnects the two devices by closing the connection. The *connectionID* is the session handle returned by `tspnet.connect()`.

For TSP-enabled devices, this aborts any remotely running commands or scripts.

Example

<pre>testID = tspnet.connect("192.0.2.0") -- Use the connection tspnet.disconnect(testID)</pre>
Create a TSP-Net session. Close the session.

Also see

[tspnet.connect\(\)](#) (on page 14-553)

tspnet.execute()

This function sends a command string to the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.execute("connectionID", "commandString")
value1 = tspnet.execute("connectionID", "commandString", formatString)
value1, value2 = tspnet.execute("connectionID", "commandString", formatString)
value1, ..., valueN = tspnet.execute("connectionID", "commandString", formatString)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>commandString</i>	The command to send to the remote device
<i>value1</i>	The first value decoded from the response message
<i>value2</i>	The second value decoded from the response message
<i>valueN</i>	The <i>N</i> th value decoded from the response message; there is one return value for each format specifier in the format string
<i>...</i>	One or more values separated with commas
<i>formatString</i>	Format string for the output

Details

This command sends a command string to the remote instrument. A termination is added to the command string when it is sent to the remote instrument (`tspnet.termination()`). You can also specify a format string, which causes the command to wait for a response from the remote instrument. The DAQ6510 decodes the response message according to the format specified in the format string and returns the message as return values from the function (see `tspnet.read()` for format specifiers).

When this command is sent to a TSP-enabled instrument, the DAQ6510 suspends operation until a timeout error is generated or until the instrument responds. The TSP prompt from the remote instrument is read and discarded. The DAQ6510 places any remotely generated errors and events into its event queue. When the optional format string is not specified, this command is equivalent to `tspnet.write()`, except that a termination is automatically added to the end of the command.

Example 1

```
tspnet.execute(instrumentID, "runScript()")
Command the remote device to run a script named runScript.
```

Example 2

```
tspnet.timeout = 5
id_instr = tspnet.connect("192.0.2.23", 23, "*rst\r\n")
tspnet.termination(id_instr, tspnet.TERM_CRLF)
tspnet.execute(id_instr, "*idn?")
print("tspnet.execute returns:", tspnet.read(id_instr))
Print the *idn? string from the remote device.
```

Also see

[tspnet.connect\(\)](#) (on page 14-553)

[tspnet.read\(\)](#) (on page 14-558)
[tspnet.termination\(\)](#) (on page 14-561)
[tspnet.write\(\)](#) (on page 14-567)

tspnet.idn()

This function retrieves the response of the remote device to *IDN?.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
idnString = tspnet.idn(connectionID)
```

<i>idnString</i>	The returned *IDN? string
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

Details

This function retrieves the response of the remote device to *IDN?.

Example

<pre>deviceID = tspnet.connect("192.0.2.1") print(tspnet.idn(deviceID)) tspnet.disconnect(deviceID)</pre>	Assume the instrument is at IP address 192.0.2.1. The output that is produced when you connect to the instrument and read the identification string may appear as: KEITHLEY INSTRUMENTS,MODEL DAQ6510,01418035,1.0.0a

Also see

[tspnet.connect\(\)](#) (on page 14-553)

tspnet.read()

This function reads data from a remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value1 = tspnet.read(connectionID)
value1 = tspnet.read(connectionID, formatString)
value1, value2 = tspnet.read(connectionID, formatString)
value1, ..., valueN = tspnet.read(connectionID, formatString)
```

<i>value1</i>	The first value decoded from the response message
<i>value2</i>	The second value decoded from the response message
<i>valueN</i>	The nth value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>formatString</i>	Format string for the output, maximum of 10 specifiers

Details

This command reads available data from the remote instrument and returns responses for the specified number of arguments.

The format string can contain the following specifiers:

%[width]s	Read data until the specified length
%[max width]t	Read data until the specified length or until punctuation is found, whichever comes first
%[max width]n	Read data until a newline or carriage return
%d	Read a number (delimited by punctuation)

A maximum of 10 format specifiers can be used for a maximum of 10 return values.

If *formatString* is not provided, the command returns a string that contains the data until a new line is reached. If no data is available, the DAQ6510 pauses operation until the requested data is available or until a timeout error is generated. Use `tspnet.timeout` to specify the timeout period.

When the DAQ6510 reads from a TSP-enabled remote instrument, the DAQ6510 removes Test Script Processor (TSP®) prompts and places any errors or events it receives from the remote instrument into its own event queue. The DAQ6510 prefaces events and errors from the remote device with `Remote Error`, followed by the event number and description.

Example

```
tspnet.write(deviceID, "*idn?\r\n")
print("write/read returns:", tspnet.read(deviceID))
```

Send the `"*idn?\r\n"` message to the instrument connected as `deviceID`.
Display the response that is read from `deviceID` (based on the `*idn?` message).

Also see

[tspnet.connect\(\)](#) (on page 14-553)

[tspnet.readavailable\(\)](#) (on page 14-559)
[tspnet.timeout](#) (on page 14-562)
[tspnet.write\(\)](#) (on page 14-567)

tspnet.readavailable()

This function checks to see if data is available from the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
bytesAvailable = tspnet.readavailable(connectionID)
```

<i>bytesAvailable</i>	The number of bytes available to be read from the connection
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

Details

This command checks to see if any output data is available from the device. No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```
ID = tspnet.connect("192.0.2.1")
tspnet.write(ID, "*idn?\r\n")
repeat bytes = tspnet.readavailable(ID) until bytes > 0
print(tspnet.read(ID))
tspnet.disconnect(ID)
```

Send commands that create data.
Wait for data to be available.

Also see

[tspnet.connect\(\)](#) (on page 14-553)
[tspnet.read\(\)](#) (on page 14-558)

tspnet.reset()

This function disconnects all TSP-Net sessions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`tspnet.reset()`

Details

This command disconnects all remote instruments connected through TSP-Net. For TSP-enabled devices, this causes any commands or scripts running remotely to be terminated.

Also see

None

tspnet.termination()

This function sets the device line termination sequence.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
type = tspnet.termination(connectionID)
type = tspnet.termination(connectionID, termSequence)
```

type	The termination type: ■ <code>tspnet.TERM_LF</code> ■ <code>tspnet.TERM_CR</code> ■ <code>tspnet.TERM_CRLF</code> ■ <code>tspnet.TERM_LFCR</code>
connectionID	The connection ID returned from <code>tspnet.connect()</code>
termSequence	The termination sequence: ■ <code>tspnet.TERM_LF</code> ■ <code>tspnet.TERM_CR</code> ■ <code>tspnet.TERM_CRLF</code> ■ <code>tspnet.TERM_LFCR</code>

Details

This function sets and gets the termination character sequence that is used to indicate the end of a line for a TSP-Net connection.

Using the *termSequence* parameter sets the termination sequence. The present termination sequence is always returned.

For the *termSequence* parameter, use the same values listed in the table above for type. There are four possible combinations, all of which are made up of line feeds (LF or 0x10) and carriage returns (CR or 0x13). For TSP-enabled devices, the default is `tspnet.TERM_LF`. For devices that are not TSP-enabled, the default is `tspnet.TERM_CRLF`.

Example

```
deviceID = tspnet.connect("192.0.2.1")
if deviceID then
    tspnet.termination(deviceID, tspnet.TERM_LF)
end
```

Sets termination type for IP address 192.0.2.1 to `TERM_LF`.

Also see

- [tspnet.connect\(\)](#) (on page 14-553)
- [tspnet.disconnect\(\)](#) (on page 14-555)

tspnet.timeout

This attribute sets the timeout value for the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	20.0 (20 s)

Usage

```
value = tspnet.timeout  
tspnet.timeout = value
```

<i>value</i>	The timeout duration in seconds: 0.001 to 30.0

Details

This attribute sets the amount of time the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands wait for a response.

The time is specified in seconds. The timeout may be specified to millisecond resolution but is only accurate to the nearest 10 ms.

Example

```
tspnet.timeout = 2.0  
Sets the timeout duration to 2 s.
```

Also see

[tspnet.connect\(\)](#) (on page 14-553)
[tspnet.execute\(\)](#) (on page 14-556)
[tspnet.read\(\)](#) (on page 14-558)

tspnet.tsp.abort()

This function causes the TSP-enabled instrument to stop executing any of the commands that were sent to it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`tspnet.tsp.abort(connectionID)`

<code>connectionID</code>	Integer value used as a handle for other <code>tspnet</code> commands

Details

This function is appropriate only for TSP-enabled instruments.
Sends an abort command to the remote instrument.

Example

<code>tspnet.tsp.abort(testConnection)</code>
Stops remote instrument execution on <code>testConnection</code> .

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Restore configuration Instrument reset Power cycle	Configuration script	1 (enable)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

value

- Enable: 1
- Disable: 0

Details

This setting determines if the instrument sends an abort message when it attempts to connect to a TSP-enabled instrument using the `tspnet.connect()` function.

When you send the abort command on an interface, it causes any other active interface on that instrument to close. If you do not send an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP-enabled remote instrument results in a connection. However, the instrument does not respond to subsequent reads or executes because control of the instrument is not obtained until an abort command has been sent.

Example

```
tspnet.tsp.abortonconnect = 0
```

Configure the instrument so that it does not send an abort command when connecting to a TSP-enabled instrument.

Also see

[tspnet.connect\(\)](#) (on page 14-553)

tspnet.tsp.rhtablecopy()

This function copies a reading buffer synchronous table from a remote instrument to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
table = tspnet.tsp.rhtablecopy(connectionID, "name")
table = tspnet.tsp.rhtablecopy(connectionID, "name", startIndex, endIndex)
```

<i>table</i>	A copy of the synchronous table or a string
<i>connectionID</i>	Integer value used as a handle for other <code>tspnet</code> commands
<i>name</i>	The full name of the reading buffer name and synchronous table to copy
<i>startIndex</i>	Integer start value
<i>endIndex</i>	Integer end value

Details

This function is only appropriate for TSP-enabled instruments.

This function reads the data from a reading buffer on a remote instrument and returns an array of numbers or a string representing the data. The *startIndex* and *endIndex* parameters specify the portion of the reading buffer to read. If no index is specified, the entire buffer is copied.

The function returns a table if the table is an array of numbers; otherwise a comma-delimited string is returned.

This command is limited to transferring 50,000 readings at a time.

Example

```
tspnet.timeout = 5
-- change the IP address in the following command
ID = tspnet.connect("134.63.79.7")
tspnet.write(ID, "login admin\r\n")
print(tspnet.read(ID))
tspnet.write(ID, "*idn?\r\n")
print(tspnet.read(ID))
print(eventlog.next())
times =
    tspnet.tsp.rhtablecopy(ID,
        "defbuffer1.timestamps", 1, 3)
print(times)
tspnet.disconnect(ID)
```

Connect to another TSP-Net enabled instrument. Copy the specified timestamps table for items 1 through 3, then display the table.

Example output:

```
SUCCESS: Logged in
KEITHLEY INSTRUMENTS,MODEL DAQ6510,04089762,1.6.3d
0 No error 0 0 0 0
05/19/2017 13:10:43.948592060,05/19/2017 13:10:44.017861380,05/19/2017
13:10:44.087080980
```

Also see

None

tspnet.tsp.runscript()

This function loads and runs a script on a remote TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

tspnet.tsp.runscript(*connectionID*, "*name*", "*script*")

<i>connectionID</i>	Integer value used as an identifier for other tspnet commands
<i>name</i>	The name that is assigned to the script
<i>script</i>	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

If you do not want the script to do anything immediately, make sure the script only defines functions for later use. Use the `tspnet.execute()` function to execute those functions later.

Example

```
tspnet.tsp.runscript(myConnection, "myTest",
"print([[start]]) for d = 1, 10 do print([[work]]) end print([[end]])")
```

Load and run a script entitled `myTest` on the TSP-enabled instrument connected with `myConnection`.

Also see

[tspnet.execute\(\)](#) (on page 14-556)

tspnet.write()

This function writes a string to the remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.write(connectionID, "inputString")
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>inputString</i>	The string to be written

Details

The `tspnet.write()` function sends *inputString* to the remote instrument. It does not wait for command completion on the remote instrument.

The DAQ6510 sends *inputString* to the remote instrument exactly as indicated. The *inputString* must contain any necessary new lines, termination, or other syntax elements needed to complete properly.

Because `tspnet.write()` does not process output from the remote instrument, do not send commands that generate too much output without processing the output. This command can stop executing if there is too much unprocessed output from previous commands.

Example

```
tspnet.write(myID, "runscript()\r\n")
```

Commands the remote instrument to execute a command or script named `runscript()` on a remote device identified in the system as *myID*.

Also see

[tspnet.connect\(\)](#) (on page 14-553)

[tspnet.read\(\)](#) (on page 14-558)

upgrade.previous()

This function returns to a previous version of the DAQ6510 firmware.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
upgrade.previous()
```


Details

This function allows you to revert to an earlier version of the firmware.

When you send this function, the instrument searches the USB flash drive in the front-panel USB port for an upgrade file. If the file is found, the instrument performs the upgrade. An error is returned if an upgrade file is not found.

If you have a communications accessory card (KTTI-GPIB, KTTI-TSP, or KTTI-RS232) installed in the instrument, the firmware on the card is also reverted to the previous version.

NOTE

Use this command with caution. Make sure your instrument can support the earlier version and that there are no compatibility issues. Check with Keithley before using this command if you have questions.

Also see

[Upgrading the firmware](#) (on page 10-7)
[upgrade.unit\(\)](#) (on page 14-568)

upgrade.unit()

This function upgrades the DAQ6510 firmware.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
upgrade.unit()
```

Details

When `upgrade.unit()` is used, the firmware is only loaded if the version of the firmware is newer than the existing version. If the version is older or at the same revision level, it is not upgraded.

When you send this function, the instrument searches the USB flash drive in the front-panel USB port for an upgrade file. If the file is found, the instrument verifies that the file is a newer version. If the version is older or at the same revision level, it is not upgraded, although it does request a reboot. If it is a newer version, the instrument performs the upgrade. An error event message is returned if no upgrade file is found.

If you have a communications accessory card (KTTI-GPIB, KTTI-TSP, or KTTI-RS232) installed in the instrument, the firmware on the card is also upgraded.

Also see

[upgrade.previous\(\)](#) (on page 14-567)
[Upgrading the firmware](#) (on page 10-7)

userstring.add()

This function adds a user-defined string to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstring.add("name", "value")
```

<i>name</i>	The name of the string; the key of the key-value pair
<i>value</i>	The string to associate with <i>name</i> ; the value of the key-value pair

Details

This function associates the string *value* with the string *name* and stores this key-value pair in nonvolatile memory.

Use the `userstring.get()` function to retrieve the *value* associated with the specified *name*.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

Example

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
for name in userstring.catalog() do
    print(name .. " = " ..
          userstring.get(name))
end
```

Stores user-defined strings in nonvolatile memory and recalls them from the instrument using a for loop.

Example output:

```
assetnumber = 236
contact = John Doe
product = Widgets
```

Also see

[userstring.catalog\(\)](#) (on page 14-570)

[userstring.delete\(\)](#) (on page 14-571)

[userstring.get\(\)](#) (on page 14-571)

userstring.catalog()

This function creates an iterator for the user-defined string catalog.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in userstring.catalog() do body end
```

name	The name of the string; the key of the key-value pair
body	Code to execute in the body of the for loop

Details

The catalog provides access for user-defined string pairs, allowing you to manipulate all the key-value pairs in nonvolatile memory. The entries are enumerated in no particular order.

Example 1

```
for name in userstring.catalog() do
  userstring.delete(name)
end
```

Deletes all user-defined strings in nonvolatile memory.

Example 2

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
for name in userstring.catalog() do
  print(name .. " = " ..
        userstring.get(name))
end
```

Prints all userstring key-value pairs.

Output:

```
product = Widgets
assetnumber = 236
contact = John Doe
```

Notice the key-value pairs are not listed in the order they were added.

Also see

- [userstring.add\(\)](#) (on page 14-569)
- [userstring.delete\(\)](#) (on page 14-571)
- [userstring.get\(\)](#) (on page 14-571)

userstring.delete()

This function deletes a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstring.delete("name")
```

<i>name</i>	The name (key) of the key-value pair of the user-defined string to delete

Details

This function deletes the string that is associated with *name* from nonvolatile memory.

Example

```
userstring.delete("assetnumber")
userstring.delete("product")
userstring.delete("contact")
```

Deletes the user-defined strings associated with the `assetnumber`, `product`, and `contact` names.

Also see

[userstring.add\(\)](#) (on page 14-569)
[userstring.catalog\(\)](#) (on page 14-570)
[userstring.get\(\)](#) (on page 14-571)

userstring.get()

This function retrieves a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value = userstring.get("name")
```

<i>value</i>	The value of the user-defined string key-value pair
<i>name</i>	The name (key) of the user-defined string

Details

This function retrieves the string that is associated with *name* from nonvolatile memory.

Example

```

userstring.add("assetnumber", "236")
value = userstring.get("assetnumber")
print(value)

```

Create the user-defined string `assetnumber`, set to a value of 236.
 Read the value associated with the user-defined string named `assetnumber`.
 Store it in a variable called `value`, then print the variable `value`.
 Output:
 236

Also see

[userstring.add\(\)](#) (on page 14-569)
[userstring.catalog\(\)](#) (on page 14-570)
[userstring.delete\(\)](#) (on page 14-571)

waitcomplete()

This function waits for all previously started overlapped commands to complete.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```

waitcomplete()
waitcomplete(group)

```

<i>group</i>	Specifies which TSP-Link group on which to wait

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `waitcomplete()` command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. This command is not needed for sequential commands.

A group number may only be specified when this node is the master node.

If no `group` is specified, the local group is used.

If zero (0) is specified for the `group`, this function waits for all nodes in the system.

NOTE

Any nodes that are not assigned to a group (group number is 0) are part of the master node group.

Example 1

```
waitcomplete()
```

Waits for all nodes in the local group.

Example 2

```
waitcomplete(G)
```

Waits for all nodes in group G.

Example 3

```
waitcomplete(0)
```

Waits for all nodes on the TSP-Link network.

Example 4

```
trigger.model.initiate()
```

```
waitcomplete()
```

Start a trigger model or scan and wait for completion.

Also see

None

Common commands

In this section:

Introduction	15-1
*CLS	15-2
*ESE	15-3
*ESR?	15-5
*IDN?	15-6
*LANG	15-8
*OPC	15-9
*RST	15-9
*SRE	15-10
*STB?	15-11
*TRG	15-11
*TST?	15-12
*WAI	15-12

Introduction

This section describes the general remote interface commands and common commands. Note that although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, the DAQ6510 does not strictly conform to that standard.

The general remote interface commands are commands that have the same general meaning, regardless of the instrument you use them with (for example, `DCL` always clears the GPIB interface and returns it to a known state).

The common commands perform operations such as reset, wait-to-continue, and status.

Common commands always begin with an asterisk (`*`) and may include one or more parameters. The command keyword is separated from the first parameter by a blank space.

If you are using a SCPI remote interface, the commands can be combined. Use a semicolon (`;`) to separate multiple commands, as shown below:

```
*RST; *CLS; *ESE 32; *OPC?
```

Although the commands in this section are shown in uppercase, they are not case sensitive (you can use either uppercase or lowercase).

If you are using the TSP remote interface, each command must be sent in a separate message.

NOTE

If you are using the TSP remote interface, note that the common commands and general bus commands cannot be used in scripts.

***CLS**

This command clears the event registers and queues.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*CLS

Details

Use the *CLS command to clear (reset to 0) the bits of the following registers in the DAQ6510:

- Operation Event Register
- Error Queue
- Questionable Event Register

It does not affect the Questionable Event Enable or Operation Event Enable registers.

This is the equivalent of sending the SCPI commands :STATus:CLEar and :SYStem:CLEar or the TSP commands `status.clear()` and `eventlog.clear()`.

To reset all the bits of the Standard Event Enable Register, send the command:

*ESE 0

Example

*CLS
Clear the event registers and queues.

Also see

- [*ESE](#) (on page 15-3)
- [:STATus:PRESet](#) (on page 12-193)
- [status.preset\(\)](#) (on page 14-431)

*ESE

This command sets and queries bits in the Status Enable register of the Standard Event Register.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	See Details

Usage

*ESE <n>
*ESE?

<n>	The value of the Status Enable register of the Standard Event Register: 0 to 255

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte Register is set on.

To set a bit on, send the constant or the value of the bit as the <n> parameter.

If you are using TSP, you can set the bit as a constant or a numeric value, as shown in the table below. To set more than one bit of the register, you can send multiple constants with + between them. You can also set *standardRegister* to the sum of their decimal weights. For example, to set bits B0 and B2, set *standardRegister* to 5 (which is the sum of 1 + 4). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.QYE
```

If you are using SCPI, you can only set the bit as a numeric value. When zero (0) is returned, no bits are set. You can also send 0 to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	status.standard.OPC	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-9) command or TSP opc() (on page 14-367) function.
1	2	Not used	Not used.
2	4	status.standard.QYE	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	status.standard.PON	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

NOTE

Constants are only available if you are using the TSP command set. If you are using the SCPI command set, you must use the decimal values.

Example

```
*ESE 129
```

*ESE 129 sets the Status Enable register of the Standard Event Register to binary 10000001, which enables the PON and OPC bits.

Also see

[*CLS](#) (on page 15-2)
[Standard Event Register](#) (on page 16-3)
[Status model](#) (on page 16-1)

*ESR?

This command reads and clears the contents of the Standard Event Status Register.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

*ESR?

Details

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register and clears the Standard Event Status Register.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	<code>status.standard.OPC</code>	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-9) command or TSP opc() (on page 14-367) function.
1	2	Not used	Not used.
2	4	<code>status.standard.QYE</code>	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Not used	Not used.
6	64	Not used	Not used.
7	128	<code>status.standard.PON</code>	The instrument has been turned off and turned back on since the last time this register was read.

Command errors include:

- **IEEE Std 488.2 syntax error:** The instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard.
- **Semantic error:** The instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented in the instrument.
- **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message.

Example

*ESR?

Example output:

128

Shows that the Standard Event Status Register contains binary 10000000, which indicates that the instrument was rebooted since the last time this register was read.

Also see

[Status model](#) (on page 16-1)

*IDN?

This command retrieves the identification string of the instrument.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

*IDN?

Details

The identification string includes the manufacturer, model number, serial number, and firmware revision of the instrument. The string is formatted as follows:

```
KEITHLEY INSTRUMENTS,MODEL nnnn,xxxxxxxx,yyyyyy
```

Where:

- *nnnn* is the model number
- *xxxxxxxx* is the serial number
- *yyyyyy* is the firmware revision

Example

```
*IDN?
```

Output:

```
KEITHLEY INSTRUMENTS,MODEL DAQ6510,01234567,1.0.0i
```

Also see

[System information](#) (on page 2-45)

*LANG

This command determines which command set is used by the instrument.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Nonvolatile memory	SCPI

Usage

```
*LANG <commandSet>
*LANG?
```

<commandSet>	The command set to be used: ■ TSP ■ SCPI ■ SCPI2700 ■ SCPI2701

Details

The remote command sets that are available include:

- **SCPI:** An instrument-specific language built on the SCPI standard.
- **TSP:** A scripting programming language that contains instrument-specific control commands that can be executed from a stand-alone instrument. You can use TSP to send individual commands or use it to combine commands into scripts.**SCPI2700:** An instrument-specific language that allows you to run code developed for Keithley Model 2700.
- **SCPI2701:** An instrument-specific language that allows you to run code developed for Keithley Model 2701.

If you change the command set, reboot the instrument.

You cannot combine the command sets.

Example

<pre>*LANG TSP *LANG?</pre>	Set the command set to TSP. Verify setting by sending the command set query. Output: TSP The TSP command set is in use.

Also see

[Status model](#) (on page 16-1)

*OPC

This command sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

*OPC
*OPC?

Details

When *OPC is sent, the OPC bit (bit 0) in the Status Event Status Register is set after all pending command operations have been executed. After all programmed operations are complete, the instrument returns to idle, at which time all pending commands (including *OPC and *OPC?) are executed. After the last pending command is executed, the OPC bit is set or an ASCII "1" is placed in the Output Queue.

When the trigger model is executing, most sent commands are not executed. If a command cannot be processed, an error event message is generated in the event log.

Example

*OPC?	If all pending OPC operations are complete, the return is 1.

Also see

[:INITiatef:IMMediate](#) (on page 12-60)
[opc\(\)](#) (on page 14-367)

*RST

This command resets the instrument settings to their default values and clears the reading buffers.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*RST

Details

Returns the instrument to default settings, cancels all pending commands, and cancels the response to any previously received *OPC and *OPC? commands.

Also see

[reset\(\)](#) (on page 14-373)

*SRE

This command sets or clears and queries the bits of Status Request Enable Register.

Type	Affected by	Where saved	Default value
Command and query	:STATus:PRESet status.preset()	Not applicable	0

Usage

*SRE <n>

*SRE?

<n>

Clear the Status Request Enable Register: 0
Set the instrument for an SRQ interrupt: 32

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	status.MSB	An enabled event in the Measurement Event Register has occurred.
1	2	Not used	Not used.
2	4	status.EAV	An error or status message is present in the Error Queue.
3	8	status.QSB	An enabled event in the Questionable Status Register has occurred.
4	16	status.MAV	A response message is present in the Output Queue.
5	32	status.ESB	An enabled event in the Standard Event Status Register has occurred.
6	64	Not used	Not used.
7	128	status.OSB	An enabled event in the Operation Status Register has occurred.

NOTE

Constants are only available if you are using the TSP command set. If you are using the SCPI command set, you must use the decimal values.

Example

*SRE 0

Clear the bits of the Status Request Enable Register.

Also see

[Understanding bit settings](#) (on page 16-18)

***STB?**

This command gets the status byte of the instrument without clearing the request service bit.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

*STB?

Details

This command is similar to a serial poll, but it is processed like any other instrument command. The *STB? command returns the same result as a serial poll, but the master summary bit (MSB) is not cleared if a serial poll has occurred. The MSB is not cleared until all other bits feeding into the MSB are cleared.

Example

*STB?	Queries the status byte.

Also see

None

***TRG**

This command generates a trigger event from a remote command interface.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*TRG

Details

Use the *TRG command to generate a trigger event. If you are using the SCPI command set, this command generates the `COMManD` event. If you are using the TSP command set, this command generates the `trigger.EVENT_COMMAND` event. You can use this constant as the stimulus of any trigger object, which causes that trigger object to respond to the trigger events generated by *TRG. See [Using trigger events to start actions in the trigger model](#) (on page 8-69).

Also see

[:INITiate\[:IMMediate\]](#) (on page 12-60)

*TST?

This command is accepted and returns 0. A self-test is not actually performed.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	0

Usage

*TST?

Also see

None

*WAI

This command postpones the execution of subsequent commands until all previous overlapped commands are finished.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

*WAI

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The *WAI command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. The *WAI command is not needed for sequential commands. Typically, this command is sent after the initiate trigger model command.

Also see

[:INITiate\[:IMMEDIATE\]](#) (on page 12-60)
[waitcomplete\(\)](#) (on page 14-572)

Status model

In this section:

Overview	16-1
Serial polling and SRQ	16-15
Programming enable registers	16-16
Reading the registers	16-17
Understanding bit settings.....	16-18
Clearing registers	16-19
Status model programming examples.....	16-20

Overview

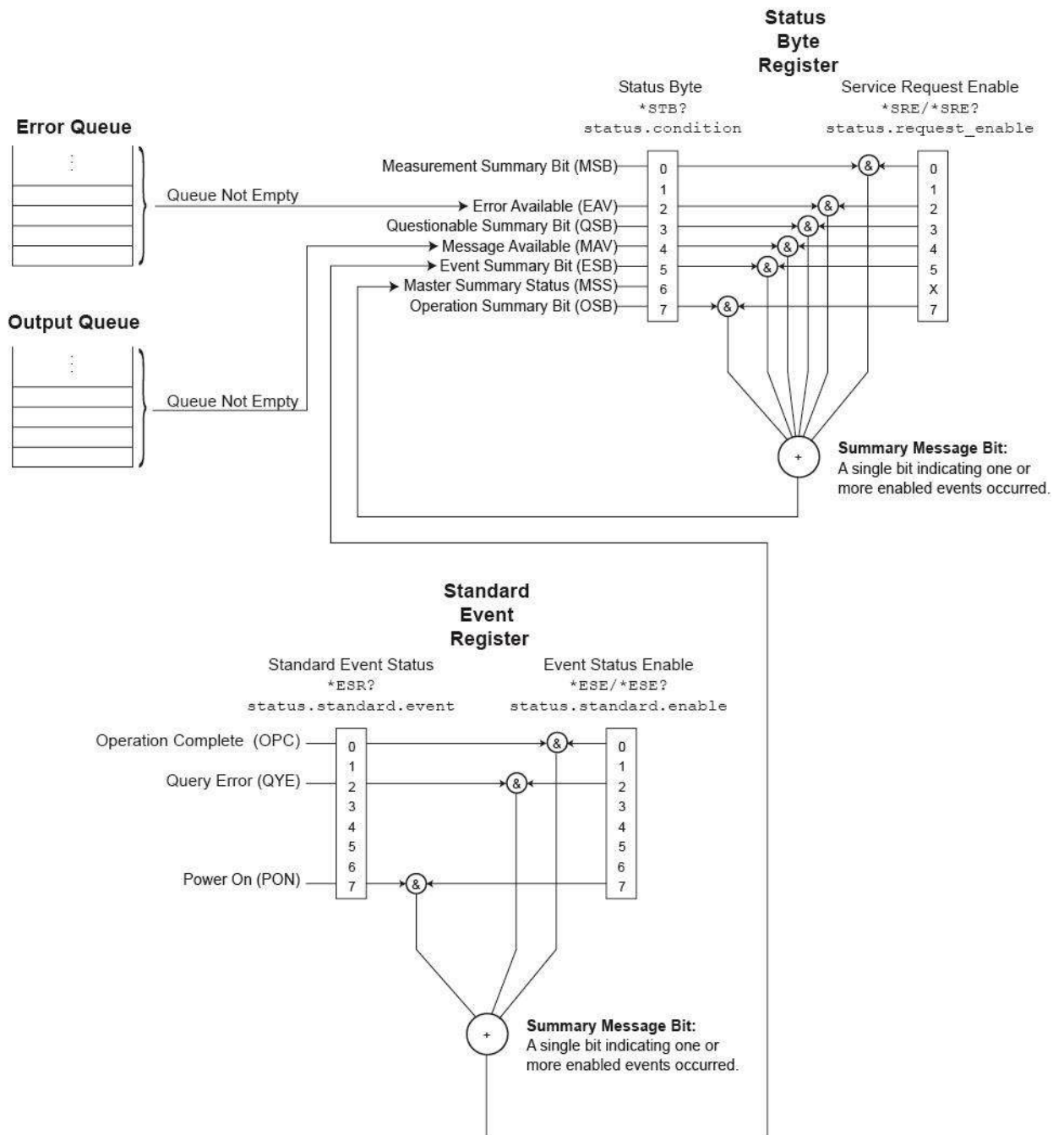
The status model consists of status register sets and queues. You can monitor the status model to view instrument events and configure the status model to control the events.

As you work with the status model, be aware that the result applies to the Status Byte Register. All the status register sets and queues flow into the Status Byte Register. Your test program can read this register to determine if a service request (SRQ) has occurred, and if so, which event caused it.

The Status Byte Register, register sets, and queues include:

- Standard Event Register
- Questionable Event Register
- Operation Event Register
- Output Queue
- Error Queue

The relationship between the Status Byte Register, Standard Event Register, event queue, and output queue is shown in the [Nonprogrammable status registers diagram](#) (on page 16-2). The relationship between the Status Byte Register, Questionable Event Register, and the Operation Event Register is shown in the [Programmable status registers diagram](#) (on page **Error! Bookmark not defined.**).

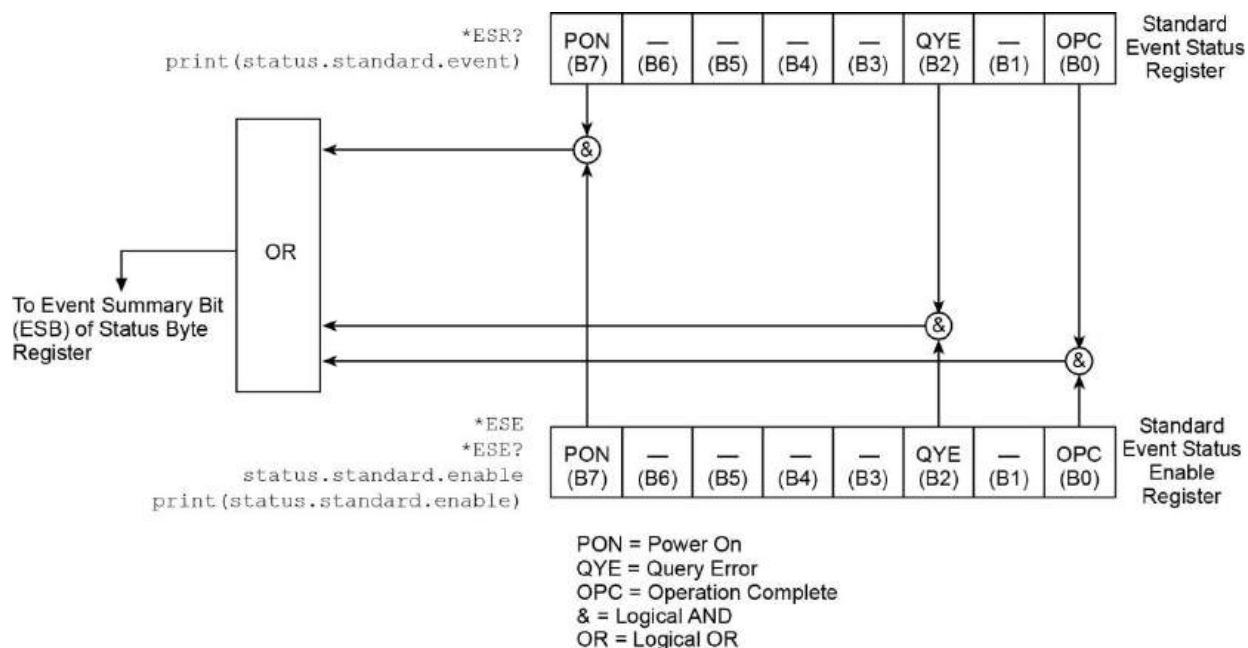
Figure 177: Nonprogrammable status registers diagram

Standard Event Register

The Standard Event Register set includes two 8-bit registers:

- **Standard Event Status Register:** Reports when a predefined event has occurred. The register latches the event and the corresponding bit remains set until it is cleared by a read.
- **Standard Event Status Enable Register:** You can enable or disable bits in this register. This allows the predefined event (from the Standard Event Status Register) to set the ESB of the Status Byte Register.

Figure 178: DAQ6510 Standard Event Register



Bit	When set, indicates the following has occurred:
0	Operation complete: All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC (on page 15-9) command or TSP opc() (on page 14-367) function.
1	Not used.
2	Query error: Attempt to read data from an empty Output Queue.
3	Not used.
4	Not used.
5	Not used.
6	Not used.
7	Power-on: The instrument has been turned off and turned back on since the last time this register was read.

You can use the following commands to read and set bits in the Standard Event Register.

Description	SCPI command	TSP command
Read the Standard Event Status Register	*ESR? (on page 15-5)	status.standard.event (on page 14-438)
Set or read the OR bits in the Standard Event Status Enable Register	*ESE (on page 15-3) ESE?	status.standard.enable (on page 14-436)

Programmable status register sets

You can program the registers in the Questionable Event Register and Operation Event Register sets.

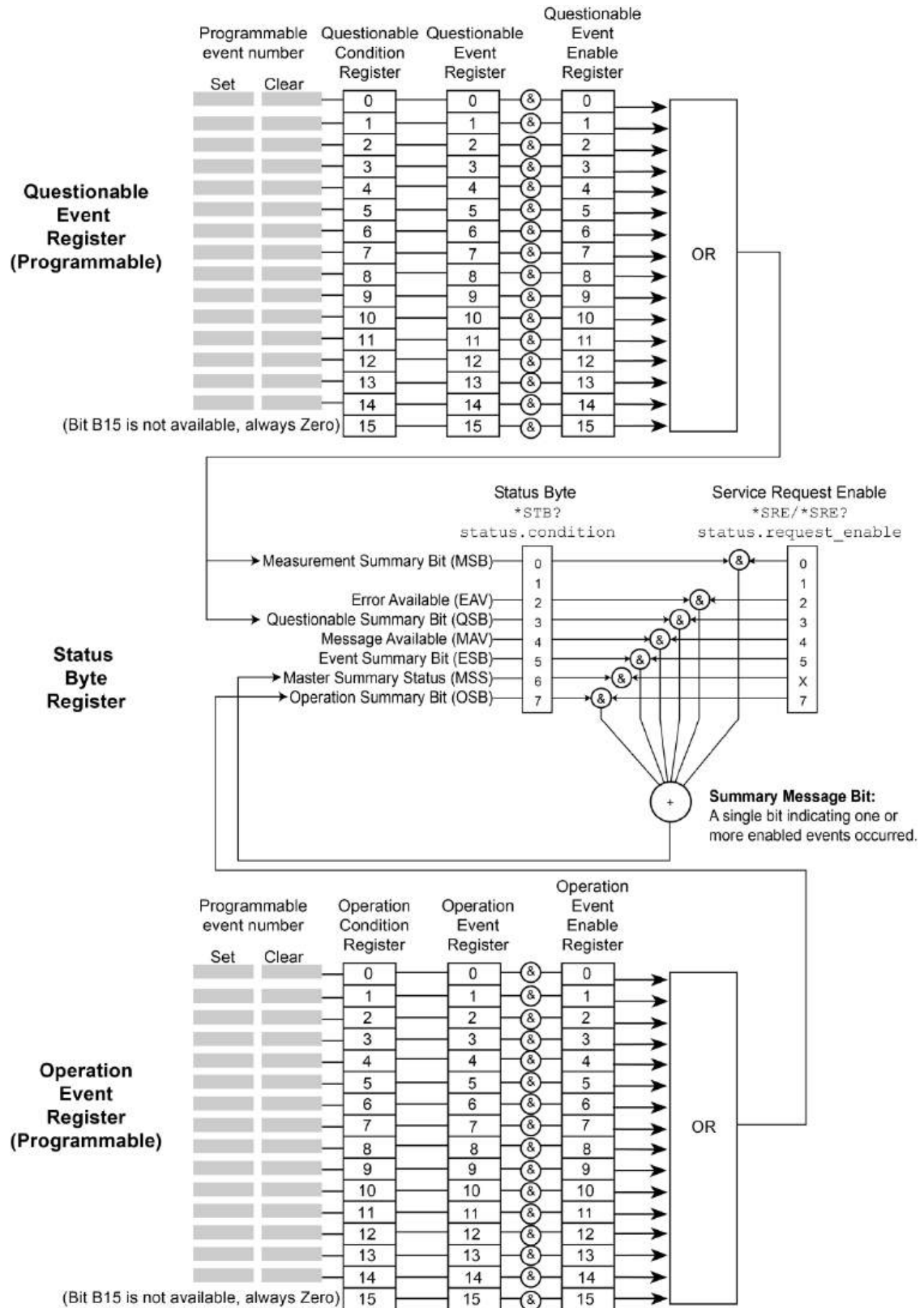
These event registers contain bits that identify the state of an instrument condition or event. They also contain bits that determine if those events are sent to the Status Byte Register. You can enable the events, which causes the associated bit to be set in the Status Byte Register.

The Questionable and Operation Event Registers are identical except that they set different bits in the Status Byte Register. The Questionable Event Registers set the MSB and QSM bits. The Operation Event Registers set the OSB bit.

Each 16-bit register set includes the following registers:

- **Condition:** A read-only register that is constantly updated to reflect the present operating conditions of the instrument. You can determine which events set or clear the bits.
- **Event:** A read-only register that sets a bit to 1 when an applicable event occurs. The bit remains at 1 until the register is reset. This register is reset when power is cycled, when a *CLS command is sent, or when the register is read. You can determine which events set the bits.
- **Event enable:** A read-write register that determines which events set the summary bit in the Status Byte Register. For example, if a bit is a 1 in the event register and the corresponding bit is a 1 in the Event Enable Register, bits in the Status Byte Register are set. If the event enable bit is set in the Questionable Event Registers, the event sets the MSB and QSM bits in the Status Byte Register. If the event enable bit is set in the Operation Event Registers, the event sets the OSB bit in the Status Byte Register.

When the instrument is powered on, all bits in the Questionable Event and Operation Event Registers are set to 0.

Figure 179: Programmable status registers diagram

Questionable Event Register

You can program the bits in the Questionable Event Register to be cleared or set when an event occurs.

When an enabled Questionable Event Register bit is set (because the enabled event occurs), the corresponding bit B0 (MSB) and Bit B3 (QSB) of the Status Byte Register is set. The corresponding Questionable Event Register Condition Register reflects the present status of the instrument, so it is set while the event occurs.

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see [Understanding bit settings](#) (on page 16-18).

You can use the following commands to read and set bits in the Questionable Event Register.

Description	SCPI command	TSP command
Read the Questionable Condition Register	:STATus:QUEStionable:CONDition? (on page 12-193)	status.questionable.condition (on page 14-431)
Set or read the contents of the Questionable Event Enable Register	:STATus:QUEStionable:ENABLE (on page 12-194)	status.questionable.enable (on page 14-432)
Read the Questionable Event Register	:STATus:QUEStionable[:EVENT]? (on page 12-195)	status.questionable.event (on page 14-432)
Request the mapped set event and mapped clear event status for a bit in the Questionable Event Register	:STATus:QUEStionable:MAP (on page 12-194)	status.questionable.getmap() (on page 14-433)
Map event to a bit in the Questionable Event Register	:STATus:QUEStionable:MAP (on page 12-194)	status.questionable.setmap() (on page 14-434)

Operation Event Register

You can program the bits in the Operation Condition and Operation Event Status Registers to be cleared or set when an event occurs.

When an enabled Operation Event Register bit is set (because the enabled event occurs), the corresponding bit B7 (OSB) of the Status Byte Register is set. The corresponding Operation Event Register Condition Register reflects the present status of the instrument, so it will be set while the event occurs.

You can use the following commands to read and set bits in the Operation Event Register.

Description	SCPI command	TSP command
Read the Operation Condition Register	:STATus:OPERation:CONDition? (on page 12-190)	status.operation.condition (on page 14-426)
Set or read the contents of the Operation Event Enable Register	:STATus:OPERation:ENABle (on page 12-191)	status.operation.enable (on page 14-427)
Read the Operation Event Register	:STATus:OPERation[:EVENTt]? (on page 12-192)	status.operation.event (on page 14-428)
Request the mapped set event and mapped clear event status for a bit in the Operation Event Registers	:STATus:OPERation:MAP (on page 12-191)	status.operation.getmap() (on page 14-429)
Map events to bit in the Operation Event Register	:STATus:OPERation:MAP (on page 12-191)	status.operation.setmap() (on page 14-430)

Mapping events to bits

To program the Questionable and Operation Event Registers, you map events to specific bits in the register. This causes a bit in the condition and event registers to be set (or cleared) when the specified event occurs. You can map events to bits B0 through B14 (bit B15 is always set to zero).

When you have a mapped-set event, the bits in the corresponding condition register and event register are set when the mapped-set event is detected. The bits remain at 1 until the event register is read or the status model is reset.

When you have a mapped-clear event, the bit in the condition register is cleared to 0 when the event is detected.

You can map any event to any bit in these registers. An event is the number that accompanies an error, warning, or informational message that is reported in the event log. For example, for the event code "Error -221, Settings Conflict," the event is -221. Note that some informational messages do not have a related event number, so they cannot be mapped to a register.

You do not need to map clear events to generate SRQs. However, if you want to read the condition register to report status, you must map both a set event and a clear event. If no clear event is mapped, the bits are cleared only when the instrument power is turned off and turned on.

You can use the following SCPI commands to read and map events to bits in the programmable registers:

- [:STATus:OPERation:MAP](#) (on page 12-191)
This command maps the set and clear events to a specified operation event register bit. Use the query form of this command to read the mapped set and clear status.
- [:STATus:QUEStionable:MAP](#) (on page 12-194)
This command maps the set and clear events to a specified operation event register bit. Use the query form of this command to read the mapped set and clear status.

You can use the following TSP commands to read and map events to bits in the programmable registers:

- [status.operation.getmap\(\)](#) (on page 14-429)
This command reads the mapped set and clear status for the specified operation event bit.
- [status.operation.setmap\(\)](#) (on page 14-430)
This command maps the set and clear events to a specified operation event register bit.

- [status.questionable.getmap\(\)](#) (on page 14-433)
This command reads the mapped set and clear status for the specified questionable event bit.
- [status.questionable.setmap\(\)](#) (on page 14-434)
This command maps the set and clear events to a specified questionable event register bit.

You can map any event that appears with a number in the event queue to any available bit in a programmable register. The programmable registers and their relationships to the Status Byte Register are shown in the [Programmable status registers diagram](#) (on page **Error! Bookmark not defined.**). The following example event queue log entries contain actual events that can be mapped to a status model bit.

```
2731 Trigger Model Initiated "Trigger model #1 has been initiated"
2732 Trigger Model Idle "Trigger model #1 has been idled"
4917 Reading buffer cleared "Reading buffer <buffer name> is 0% filled"
4918 Reading buffer full "Reading buffer <buffer name> is 100% filled"
```

See [Using the event log](#) (on page 3-80) for additional information on finding events.

Status Byte Register

The Status Byte Register monitors the registers and queues in the status model and generates service requests (SRQs).

When bits are set in the status model registers and queues, they generate summary messages that set or clear bits of the Status Byte Register. You can enable these bits to generate an SRQ.

Service requests (SRQs) instruct the controller that the instrument needs attention or that some event has occurred. When the controller receives an SRQ, the controller can interrupt existing tasks to perform tasks that address the request for service.

For example, you might program your instrument to send an SRQ when a specific instrument error event occurs. To do this, you set the Status Request Enable bit 2 (EAV). In this example, the following actions occur:

- The error event occurs.
- The error event is logged in the Error Queue.
- The Error Queue sets the EAV bit of the Status Byte Register.
- The EAV bits are summed.
- The RQS bit of the Status Byte Register is set.
- On a GPIB system, the SRQ line is asserted. On a VXI-11 or USB system, an SRQ event is generated.

For an example of this, see the example code provided in [SRQ on error](#) (on page 16-20).

The summary messages from the status registers and queues set or clear the appropriate bits (B0, B2, B3, B4, B5, and B7) of the Status Byte Register. These summary bits do not latch, and their states (0 or 1) are solely dependent on the summary messages (0 or 1). For example, if the Standard Event Register is read, its register will clear. As a result, its summary message resets to 0, which in turn resets the ESB bit in the Status Byte Register.

The Status Byte Register also receives summary bits from itself, which sets the Master Summary Status (MSS) bit.

When using the GPIB, USB, or VXI-11 serial poll sequence of the DAQ6510 to get the status byte (serial poll byte), bit B6 is the RQS bit. See [Serial polling and SRQ](#) (on page 16-15) for details on using the serial poll sequence.

When using the `*STB?` common command or `status.condition` command to read the status byte, bit B6 is the MSS bit.

To reset the bits of the Service Request Enable Register to 0, use 0 as the parameter value for the command (for example, `*SRE 0` or `status.request_enable = 0`).

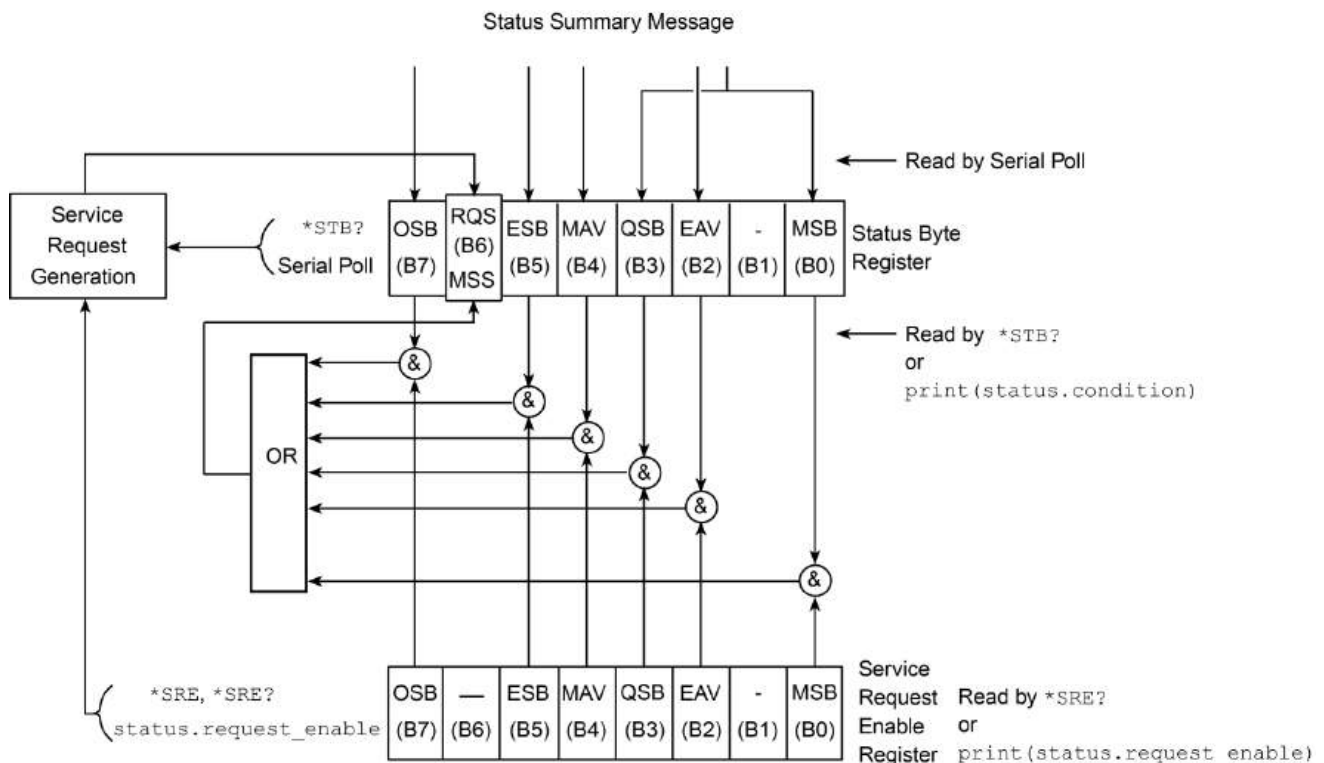
You can read and set which bits to AND in the Status Byte Register using the following commands.

Description	SCPI command	TSP command
Read the Status Byte Register	<code>*STB?</code> (on page 15-11)	status.condition (on page 14-424)
Read the Status Request Enable Register	<code>*SRE</code> (on page 15-10)	status.request_enable (on page 14-435)
Enable bits in the Status Request Enable Register	<code>*SRE</code> (on page 15-10)	status.request_enable (on page 14-435)

Status Byte Register diagram

The Status Byte Register consists of two 8-bit registers that control service requests, the Status Byte Register and the Service Request Enable Register. These registers are shown in the following figure.

Figure 180: DAQ6510 Status Byte Register



The bits in the Status Byte Register are described in the following table.

Bit	Decimal value	Bit name	When set, indicates the following has occurred:
0	1	Measurement summary Bit (MSB)	An enabled questionable event
1	2	Not used	Not applicable
2	4	Error available (EAV)	An error is present in the error queue (warning and information messages do not affect this bit)
3	8	Questionable summary bit (QSB)	An enabled questionable event
4	16	Message available (MAV)	A response message is present in the output queue
5	32	Event summary bit (ESB)	An enabled standard event
6	64	Request for service (RQS)/Master summary status (MSS)	An enabled summary bit of the Status Byte Register is set; depending on how it is used, this is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit
7	128	Operation summary bit (OSB)	An enabled operation event

Service Request Enable Register

This register is programmed by the user and is used to enable or disable the setting of bit B6 (RQS/MSS) by the Status Summary Message bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. As shown in the [Status Byte Register](#) (on page 16-11) topic, a logical AND operation is performed on the summary bits (&) with the corresponding enable bits of the Service Request Enable Register. When a logical AND operation is performed with a set summary bit (1) and with an enabled bit (1) of the enable register, the logic “1” output is applied to the input of the logical OR gate and, therefore, sets the MSS/RQS bit in the Status Byte Register.

You can set or clear the individual bits of the Service Request Enable Register by using the `*SRE` common command or `status.request_enable`. To read the Service Request Enable Register, use the `*SRE?` query or `print(status.request_enable)`. The Service Request Enable Register clears when power is cycled or a parameter value of 0 is sent with a status request enable command (for example, a `*SRE 0` or `status.request_enable = 0` is sent). You can program and read the SRQ Enable Register using the following commands.

Description	SCPI command	TSP command
Read the Status Request Enable Register	*SRE (on page 15-10)	status.request_enable (on page 14-435)
Enable bits in the Status Request Enable Register	*SRE (on page 15-10)	status.request_enable (on page 14-435)

Queues

The instrument includes an Output Queue and an Error Queue. The Output Queue holds messages from readings and responses. The Error Queue holds error event messages from the event log. Both are first-in, first-out (FIFO) registers.

Output Queue

The output queue holds response messages to SCPI query and TSP `print()` commands.

When data is placed in the Output Queue, the Message Available (MAV) bit in the Status Byte Register is set. The bit is cleared when the Output Queue is empty.

To clear data from the Output Queue, read the messages. To read a message from the Output Queue, address the instrument to talk after the appropriate query is sent.

Error Queue

The Error Queue holds any error events that are posted in the event log. When an error event occurs, it is posted to the Error Queue, which sets the Error Available (EAV) bit in the Status Byte Register.

The instrument clears error event messages from the event log when it retrieves the event log. When the error event messages are cleared from the event log, the EAV bit in the Status Byte Register is cleared.

You can clear the Error Queue by sending the common command `*CLS` or the TSP command `status.clear()`. Note that `status.clear()` also clears all event registers.

For information regarding the event log, see [Using the event log](#) (on page 3-80).

Serial polling and SRQ

Any enabled event summary bit that goes from 0 to 1 sets bit B6 and generates a service request (SRQ).

In your test program, you can periodically read the Status Byte to check if an SRQ occurred and what caused it. If an SRQ occurred, the program can, for example, branch to an appropriate subroutine that services the request.

SRQs can be managed by the serial poll sequence of the instrument. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register remains cleared, and the program proceeds normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register is set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte Register. This allows subsequent serial polls to monitor bit B6 for an SRQ occurrence that is generated by other event types.

The serial poll does not clear the low-level registers that caused the SRQ to occur. You must clear the low-level registers explicitly. Refer to [Clearing registers](#) (on page 16-19).

For common commands and TSP commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear the MSS bit. The MSS bit remains set until all enabled Status Byte Register summary bits are reset.

Programming enable registers

You can program the bits in the enable registers of the Status Model registers.

When you program an enable register bit to 0, no action occurs if the bits in the corresponding registers are set (1).

When you program an enable register bit to 1, if the bits in the corresponding registers are set (1), the AND condition occurs and a bit in the Status Byte Register is set to (1).

You must program all bits in an enable register at the same time. This means you need to determine what each bit value in the register will be, then add them together to determine the value of all the bits in the register. See [Understanding bit settings](#) (on page 16-18) for more information on determining the value of the bits in the registers.

For example, you might want to enable the Standard Event Register to set the ESB bit in the Status Byte Register whenever an operation complete occurs or whenever an operation did not execute properly because of an internal condition. To do this, you need to set bits 0 and 3 of the Standard Event Register to 1. These bits have decimal values of 1 and 8, so to set both bits to 1, you set the register to 9.

Using SCPI:

Send the command:

```
*ese 9
```

Using TSP

Send the command:

```
status.standard.enable = 9
```

Reading the registers

You can read any register in the status model. The response is a decimal value that indicates which bits in the register are set. See [Understanding bit settings](#) (on page 16-18) for information on how to convert the decimal value to bits.

Using SCPI commands:

If you are using SCPI, you use the query commands in the STATUS subsystem and common commands to read registers.

Using TSP commands:

If you are using TSP, you print the TSP command to read the register. You can use either `print()`, which returns the decimal value, or `print(tostring())`, which returns the string equivalent of the decimal value.

You can also send the common commands to read the register.

For example, you can send any one of the following commands to read the Status Enable Register of the Standard Event Register:

```
print(status.standard.enable)
*ese?
print(tostring(status.standard.enable))
```

Understanding bit settings

When you write to or read a status register, you can use binary, decimal, or hexadecimal values to represent the binary values of the bit states. When the value is converted to its binary equivalent, you can determine which bits are set on or clear. Zero (0) indicates that all bits are clear.

In the DAQ6510, the least significant bit is always bit B0. The most significant bit differs for each register, but in most cases is either bit B7 or bit B15.

Bit position	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	1000 0000	0100 0000	0010 0000	0001 0000	1000	0100	0010	0001
Decimal value	128	64	32	16	8	4	2	1
Weight	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Bit position	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	1000 0000 0000 0000	0100 0000 0000 0000	0010 0000 0000 0000	0001 0000 0000 0000	1000 0000 0000 0000	0100 0000 0000 0000	0010 0000 0000 0000	0001 0000 0000 0000
Decimal value	32768	16384	8192	4096	2048	1024	512	256
Weight	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8

For example, if a value of 129 is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set and all other bits are cleared. If you read a value of 12288 for the condition register, the binary equivalent is 0011 0000 0000 0000. This value indicates that bits B12 and B13 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

When bit B12 (4096) and bit B13 (8192) are set (1), the decimal equivalent is $4096 + 8192 = 12,288$.

Clearing registers

Registers in the status model can be cleared using commands or by instrument actions. When a register is cleared, the bits in the register are set to 0.

The event log and all registers are cleared when instrument power is cycled.

When you read a bit from the Operation Event, Questionable Event, or Standard Event Status Register, the entire 16-bit or 8-bit register value is returned. The event register is cleared or set to 0.

Using SCPI commands:

To clear the event registers of the Questionable Event Status Register and Operation Event Status Register sets, Standard Event Register, and Status Byte Register, send:

```
*CLS
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register and Operation Event Enable Register sets.

To clear the Questionable Event Status Register, the Operation Event Status Register sets, Standard Event Register, and Status Byte Register, send:

```
STATus:CLEar
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register or Operation Event Enable Register sets.

Using TSP commands:

To clear the event registers of the Questionable Event Status Register and Operation Event Status Register sets, Standard Event Register, and Status Byte Register send:

```
*CLS
```

To clear the Questionable Event Status Register, the Operation Event Status Register sets, Standard Event Register, and Status Byte Register send:

```
status.clear()
```

When using the SCPI interface, this command does not affect the Questionable Event Enable Register or Operation Event Enable Register sets.

Status model programming examples

The following examples illustrate how to generate an SRQ using the status model.

SRQ on error

This example shows you how to generate a service request (SRQ) when an instrument error event occurs.

Using SCPI commands:

```
*RST
SYST:CLE
STAT:CLE
*SRE 4
MAKEERROR
```

Using TSP commands:

```
reset()
-- Clear Error Queue so EAV bit can go low.
eventlog.clear()

-- Clear the status byte.
status.clear()

-- Enable SRQ on error available.
status.request_enable = status.EAV

-- Send a line of code that will generate an error event.
beeper = 1
```

SRQ when reading buffer becomes full

This example shows you how to generate a service request (SRQ) when the DAQ6510 reading buffer is full. You can use this to notify the controlling computer that it needs to read back the data and empty the buffer. After configuring the status model, this code configures the default reading buffer 1 to a size of 100, and then configures the DAQ6510 to fill the buffer. After the buffer is full, the instrument generates an SRQ and returns the data.

Using SCPI commands:

```
*RST
STAT:CLE
STAT:OPER:MAP 0, 4918, 4917
STAT:OPER:ENAB 1
*SRE 128
TRAC:CLE
TRAC:POIN 100, "defbuffer1"
COUNT 100
READ? "defbuffer1"
TRAC:DATA? 1, 100, "defbuffer1", READ
```

Using TSP commands:

```
reset()
-- Clear the status byte
status.clear()

-- Map bit 0 of operational status register to set on default buffer
-- full (4918) and clear on buffer empty (4917).
status.operation.setmap(0, 4918, 4917)

-- Enable bit 0 to flow through to the status byte.
status.operation.enable = 1

-- Enable the Operational Summary Bit to set the Master
-- Summary Bit/RQS
status.request_enable = status.OSB

-- Clear the buffer and make it smaller
defbuffer1.clear()
defbuffer1.capacity = 100

-- Set the measure count to fill the buffer
dmm.measure.count = 100
dmm.measure.range = 10e-3
dmm.measure.read(defbuffer1)

printbuffer(1, defbuffer1.n, defbuffer1)
```

Frequently asked questions

In this section:

How do I save the present state of the instrument?	17-2
What is the ethernet port number?	17-2
What to do if the GPIB controller is not recognized?	17-3
I am receiving GPIB timeout errors. What should I do?	17-3
Can I use Keysight GPIB cards with Keithley drivers?	17-3
What does -113 "Undefined header" error mean?	17-4
What does -410 "Query interrupted" error mean?	17-4
What does -420 "Query unterminated" error mean?	17-5
What is error 5503?	17-5
How do I upgrade the firmware?	17-5
Where can I find updated drivers?	17-6
Why did my settings change?	17-7
Why can't the DAQ6510 read my USB flash drive?	17-7
How do I check the USB driver for the device?	17-7
Why are there dashes on the front-panel display?	17-9
How do I display the instrument's serial number?	17-10
Why doesn't the DAQ6510 recognize my switch card?	17-11
Where is the current fuse located?	17-11
Where can I find the EU Declaration of Conformity?	17-11
What VISA resource name is required?	17-12
How do I control the backplane channels using multiple close commands?	17-12
Can you program the TERMINALS switch?	17-13
Are the channels on the switching module isolated from earth ground?	17-13
Getting readings on scale in four-wire ohms.	17-13
Is the thermistor sensor for temperature two-wire or four-wire?	17-13
What is offset compensation?	17-13
What are the test current values for the resistance ranges?	17-14
What are the current shunt resistors in the DAQ6510?	17-14
What happens when thermocouple open detect is enabled?	17-14
What is a configuration list?	17-15
How do I change the command set?	17-15
How do I obtain the fastest measurement rate?	17-16
Can I close one channel and keep it closed while other channels are scanned?	17-17
I see a command that is not in the manual. What is it?	17-18
Can the DAQ6510 have multiple user-defined RTDs in a scan list?	17-18
What rack kit is required for mounting?	17-18
Which rack kit is required to rack mount a DAQ6510 and 2280 side-by-side?	17-18

How do I save the present state of the instrument?

You can save the settings in the instrument as a script using the front-panel menus or from a remote interface. After they are saved, you can recall the script or copy it to a USB flash drive.

From the front panel:

1. Configure the DAQ6510 to the settings that you want to save.
2. Press the **MENU** key.
3. Under Scripts, select **Save Setup**.
4. Select **Create**. A keyboard is displayed.
5. Use the keyboard to enter the name of the script.
6. Select the **OK** button on the displayed keyboard. The script is added to internal memory.

Using SCPI commands:

1. Configure the instrument to the settings that you want to save.
2. Send the command:

`*SAV <n>`

Where <n> is an integer from 0 to 4.

NOTE

In the front-panel script menus, the setups saved with the `*SAV` command have the name `Setup0x`, where `x` is the value you set for <n>.

Using TSP commands:

1. Configure the instrument to the settings that you want to save.
2. Send the command:
`createconfigscript("setupName")`
Where `setupName` is the name of the setup script that is created.

What is the ethernet port number?

5025

What to do if the GPIB controller is not recognized?

If the hardware is not recognized by the computer:

1. Check the website of the vendor for newer drivers.
2. Check that the drivers are valid for the operating system you have and any updates that might be necessary. This information is typically found in the readme file that comes with the drivers.
3. Follow the instructions from the vendor on updating drivers.

If the hardware is still not recognized, you can try a different computer using a different operating system to rule out operating system issues.

If this does not resolve the issue, contact the vendor of the GPIB controller for assistance.

I am receiving GPIB timeout errors. What should I do?

If your GPIB controller is recognized by the operating system, but you get a timeout error when you try to communicate with the instrument, check the following:

1. Confirm that the GPIB address you assigned to the instrument is unique and between 0 to 30. Do not use 0 or 21 because they are common controller addresses.
2. Check cabling connections. GPIB cables are heavy and can fall out of the connectors if they are not screwed in securely.
3. Substitute cables to verify cable integrity. For example, if you can send and receive ASCII text, but you cannot do a binary transfer, check your program and the decoding of the binary data. If that does not resolve the problem, try another cable. ASCII text only uses seven data lines in the cable; the binary transfer requires all eight lines.

Can I use Keysight GPIB cards with Keithley drivers?

Yes, if the instrument driver uses VISA for instrument communications. This is true for any instrument driver that is IVI or VXI/PnP based.

What does –113 "Undefined header" error mean?

When using the SCPI command language, you may see the –113, "Undefined header," error. This error indicates that what you sent to the instrument did not contain a recognizable command name. The most likely causes for this are:

- A missing space between the command and its parameter. There must be one or more spaces between the command and its parameter. For example,

```
:disp:volt:digits5
```

The correct entry is

```
:disp:volt:digits 5
```

- Incorrect short or long form. Check the [SCPI command reference](#) (on page 12-1) documentation for the correct command name.
- Spaces in the command name. You cannot use spaces in the command name. For example:

```
syst: err?
```

The correct entry is:

```
:syst:err?
```

What does –410 "Query interrupted" error mean?

This error occurs when you have sent a valid query to the instrument and then send it another command, query, or a Group Execute Trigger (GET) before it has had a chance to send the entire response message (including the line-feed/EOI terminator). The most likely causes are:

- Sending a query to the instrument and then sending another command or query before reading the response to the first query. For example, the following sequence of commands causes an error -410:

```
syst:err?
```

```
*opc?
```

You must read the response to `syst:err?` before sending another command or query.

- Incorrectly configured IEEE 488 driver. The driver must be configured so that when talking on the bus it sends line-feed with EOI as the terminator, and when listening on the bus it expects line-feed with EOI as the terminator. See the reference manual for your IEEE 488 interface.

What does -420 "Query unterminated" error mean?

This error occurs when you address the instrument to talk and it has nothing to say. The most likely causes are:

- A query was not sent. You must send a valid query to the instrument before addressing it to talk. You cannot get a reading until you send the instrument a query.
- An invalid query was sent. If you sent a query and get this error, make sure that the instrument is processing the query without error. For example, sending a query that generates an "Undefined header" error and then addressing the instrument to talk will generate a "Query unterminated" error.
- A valid query in a command string that also contains an invalid command. This can occur when you send multiple commands or queries in one command string (program message). When the instrument detects an error in a command string, it discards all further commands in the command string until the end of the string. For example, this command string would result in a query unterminated error:
`:sens:date?;;sens:func?`

The first command (`:sens:date?`) generates error -113, "Undefined header" and the instrument discards the second command (`:sens:func?`), even though it is a valid query.

What is error 5503?

Error 5503 means the card relay counts are lost.

How do I upgrade the firmware?

CAUTION

Do not turn off power or remove the USB flash drive until the upgrade process is complete.

CAUTION

Do not initialize TSP-Link before starting the upgrade.

NOTE

You can upgrade or downgrade the firmware from the front panel.

From the front panel:

1. Copy the firmware file (.upg file) to a USB flash drive.
2. Verify that the firmware file is in the root subdirectory of the flash drive and that it is the only firmware file in that location.
3. Disconnect any terminals that are attached to the instrument.
4. Turn the instrument power off. Wait a few seconds.
5. Turn the instrument power on.
6. Insert the flash drive into the USB port on the front panel of the instrument.
7. From the instrument front panel, press the **MENU** key.
8. Under System, select **Info/Manage**.
9. Choose an upgrade option:
 - To upgrade to a newer version of firmware, select **Upgrade to New**.
 - To return to a previous version of firmware, select **Downgrade to Older**.
10. If the instrument is controlled remotely, a message is displayed. Select **Yes** to continue.
11. When the upgrade is complete, reboot the instrument.

A message is displayed while the upgrade is in progress.

For additional information about upgrading the firmware, see [Upgrading the firmware](#) (on page 10-7).

Where can I find updated drivers?

For the latest drivers and additional support information, see tek.com/support.

To see what drivers are available for your instrument:

1. Go to tek.com/support.
2. Select Product Support & Downloads.
3. Enter the model number of your instrument.
4. Select **Software** from the filter list.
5. Select **Driver** from the filter list.

NOTE

If you use the native LabVIEW™ or IVI driver, you must configure the DAQ6510 to use the SCPI command set. For information on changing the command set, refer to [How do I change the command set?](#) (on page 17-15).

Why did my settings change?

Many of the commands in the DAQ6510 are saved with the measure function that was active when you set them. For example, assume you have the measure function set to current and you set a value for display digits. When you change the measure function to voltage, the display digits value changes to the value that was last set for the voltage measure function. When you return to the current measure function, the display digits value returns to the value you set previously.

Why can't the DAQ6510 read my USB flash drive?

Verify that the flash drive is formatted with the FAT32 file system. The DAQ6510 only supports FAT and FAT32 drives using Master Boot Record (MBR).

In Microsoft® Windows®, you can check the file system by checking the properties of the USB flash drive.

NOTE

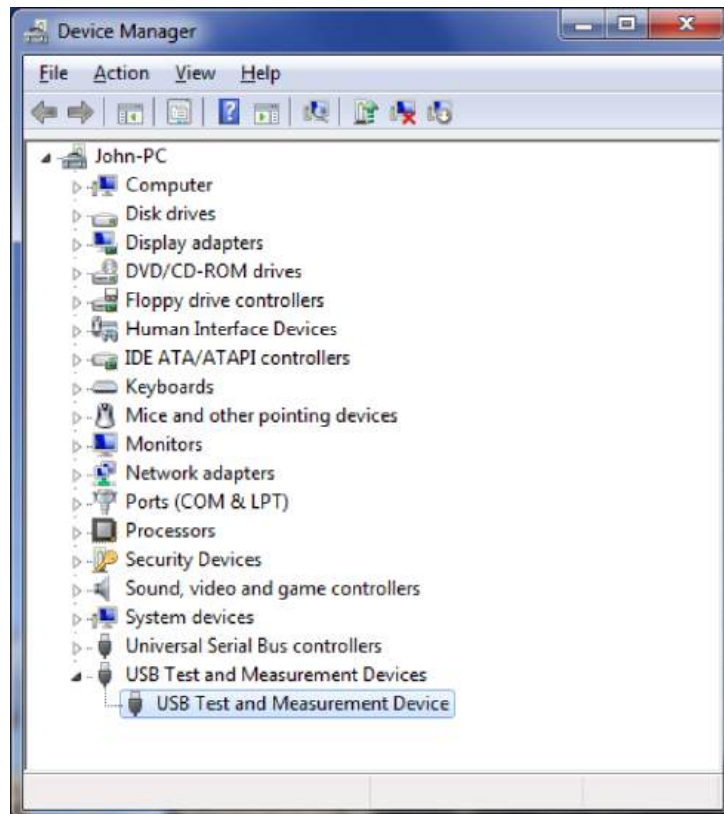
Higher capacity USB drives take longer to be read and loaded by the instrument.

How do I check the USB driver for the device?

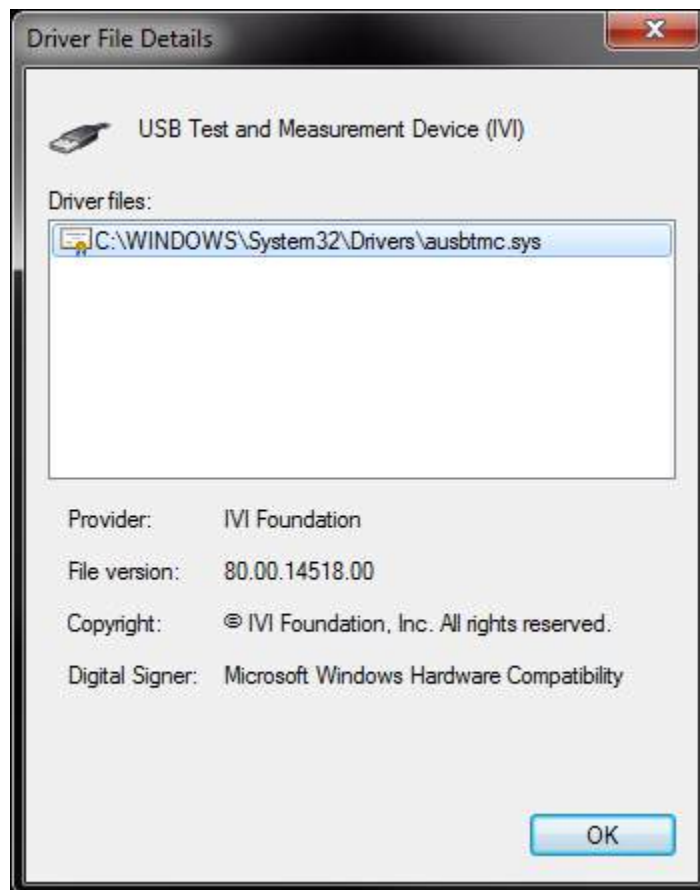
To check the driver for the USB Test and Measurement Device:

1. Open Device Manager. From the **Start** menu, you can enter `devmgmt.msc` in the Run box or the Windows search box to start Device Manager.
2. Under USB Test and Measurement Devices, look for USB Test and Measurement Device.

If the device is not there, either VISA is not installed or the instrument is not plugged in and switched on.

Figure 181: Device Manager dialog box showing USB Test and Measurement Device

3. Right-click the device.
4. Select **Properties**.
5. Select the **Driver** tab.
6. Select **Driver Details**.
7. Verify that the device driver is the `ausbtmc.sys` driver from IVI Foundation.

Figure 182: Driver File Details dialog box

8. If the incorrect driver is installed, select **OK**.

If this does not work, uninstall VISA, unplug the instrument, and follow the steps to reinstall VISA in the section [Modifying, repairing, or removing Keithley I/O Layer software](#) (on page 2-40).

Why are there dashes on the front-panel display?

There is no previous measurement for the present function.

How do I display the instrument's serial number?

The instrument serial number is on a label on the rear panel of the instrument. You can also access the serial number from the front panel or by using remote commands.

To view the system information from the front panel:

1. Press the **MENU** key.
2. Under System, select **Info/Manage**. The system information displays, including the serial number.
3. To return to the home screen, select the **HOME** key.

To view system information using SCPI commands:

Send the command:

```
*IDN?
```

To view system information using TSP commands:

Send the command:

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

Why doesn't the DAQ6510 recognize my switch card?

Card is not correctly seated in the slot

Make sure the card is seated correctly:

1. Remove power from the instrument.
2. Pull the card out of the slot.
3. Carefully guide the card inside the rails while reinstalling.
4. Make sure the outer shield is not inserted into the card rails.
5. Make sure that no other portions of the card catch on any part of the instrument frame while installing.
6. Make sure the card is seated correctly before restarting the instrument.

Edge connector is dirty

If the card was stored outside of the instrument, it is possible that the edge connector is dirty.

1. Check the edge connector. The gold edge connector fingers should have a bright surface when properly cleaned.
2. If necessary, clean the edge connector.
3. Reinstall the card.

Confirm that the card is supported in the DAQ6510

See the DAQ6510 specifications.

Where is the current fuse located?

The current fuse is in the white AMPS terminal. To replace the fuse, refer to [Current input fuse replacement](#) (on page 10-4).

Where can I find the EU Declaration of Conformity?

Please contact Keithley Instruments to get a copy of the EU Declaration of Conformity for this product.

What VISA resource name is required?

To determine the VISA resource name that is required to communicate with the instrument, you can run the Keithley Configuration Panel. The Configuration Panel automatically detects all instruments connected to the computer.

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

To run the Configuration Panel, select **Start > All Programs > Keithley Instruments > Keithley Configuration Panel** and follow the steps in the wizard.

How do I control the backplane channels using multiple close commands?

When using the independent channel control commands to configure multiple channels when measuring, you must also close the backplane relays.

For example:

Using SCPI commands:

To close channels 101 and 107 with no measurement, send the command:

```
ROUTe:MULTiple:CLOSe (@101,107)
```

If the measurement is required, channel 125 on the 20-channel switching module or channel 145 on the 40-channel switching module must also be closed.

Using SCPI commands:

To close channel 101 and 125, send the command:

```
ROUTe:MULTiple:CLOSe (@101,125)
```

Can you program the TERMINALS switch?

No, you cannot set the FRONT/REAR TERMINALS switch remotely. It is a mechanical switch. You can read the status of the switch remotely by using the following commands.

To view system information using SCPI commands, send the command:

```
ROUTe:TERMinals?
```

To view system information using TSP commands, send the command:

```
print(dmm.terminals)
```

Are the channels on the switching module isolated from earth ground?

Yes. They are isolated from earth ground. The common mode voltage is 300.

Getting readings on scale in four-wire ohms.

Check the connections to the measured device.

Check connections to make sure it is connected to the device under test Source HI, Source LO, Sense HI, and Sense LO. If they are properly connected, check the Terminals switch on the front panel. Make sure it is set to the set of terminals you are using to test the resistance.

Is the thermistor sensor for temperature two-wire or four-wire?

Thermistors in general are two-wire. On the switching module, it is two-wire.

What is offset compensation?

Offset compensation is a measuring technique that reduces or eliminates electric and magnetic fields (EMFs) in low-level resistance measurements. The voltage offsets that occur because of the presence of V_{EMF} can adversely affect resistance measurement accuracy.

To overcome these offset voltages, you can use offset-compensated ohms.

What are the test current values for the resistance ranges?

- 1 Ω range = 10 mA
- 10 Ω range = 10 mA
- 100 Ω range = 1 mA
- 1 k Ω range = 1 mA
- 10 k Ω range = 100 μ A
- 100 k Ω range = 10 μ A
- 1 M Ω range = 10 μ A
- 10 M Ω range = 0.7 μ A in parallel with a 10 M Ω reference resistor (using the ratiometric method)
- 100 M Ω range = 0.7 μ A in parallel with a 10 M Ω reference resistor (using the ratiometric method)

What are the current shunt resistors in the DAQ6510?

They are:

- 10 μ A: 10 k Ω
- 100 μ A: 1 k Ω
- 1 mA: 100 Ω
- 10 mA: 10 Ω
- 100 mA: 1 Ω
- 1 A: 100 m Ω
- 3 A: 100 m Ω

What happens when thermocouple open detect is enabled?

The DAQ6510 has an open thermocouple detection circuit. When enabled, a 100 μ A pulse of current is applied to the thermocouple before the start of each temperature measurement. If >1.2 k Ω is detected (open thermocouple), the OVRFLW message is displayed. If <1.2 k Ω is detected, the current is turned off and a normal thermocouple temperature measurement is performed.

What is a configuration list?

A configuration list is a list of stored instrument settings. You can restore these instrument settings to change the active state of the instrument. Configuration lists allow you to record the active state of the instrument, store it, and then return the instrument to that state as needed.

If you are using TSP, configuration lists run faster than a script that is set up to configure the same settings.

The DAQ6510 supports measure configuration lists, making it possible to sequence through defined measure settings.

Each configuration list consists of a list of configuration indexes. A configuration index contains all instrument measure settings that were active at a specific point. You can cycle through the configuration indexes using a trigger model.

For more detail, see [Configuration lists](#) (on page 4-99).

How do I change the command set?

You can change the command set that you use with the DAQ6510. The remote command sets that are available include:

- **SCPI:** An instrument-specific language built on the SCPI standard.
- **TSP:** A scripting programming language that contains instrument-specific control commands that can be executed from a stand-alone instrument. You can use TSP to send individual commands or use it to combine commands into scripts.
- **SCPI2700:** An instrument-specific language that allows you to run code developed for Keithley Model 2700.
- **SCPI2701:** An instrument-specific language that allows you to run code developed for Keithley Model 2701.

If you change the command set, reboot the instrument.

You cannot combine the command sets.

NOTE

As delivered from Keithley, the DAQ6510 is set to work with the SCPI command set.

NOTE

If you choose the SCPI2700 or SCPI2701 command set, you will not have access to some of the extended ranges and other features that are now available using the default SCPI command set. In addition, some Series 2700 code works differently in the DAQ6510 than it did in the earlier instrument. See *DAQ6510 in a Model 2700 or 2701 Application*, Keithley document number 0771465XX, for information about the differences.

To set the command set from the front panel:

1. Press the **MENU** key.
2. Under System, select **Settings**.
3. Select the appropriate **Command Set**.

You are prompted to confirm the change to the command set and reboot the instrument.

To verify which command set is selected from a remote interface, send the command:

```
*LANG?
```

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

```
*LANG SCPI
```

Reboot the instrument.

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

```
*LANG TSP
```

Reboot the instrument.

How do I obtain the fastest measurement rate?

Use the Performance slider to adjust for performance (resolution versus speed).

When you adjust the Performance slider, the instrument changes settings based on where you position the slider. As you increase reading speed, you lower the amount of resolution. As you increase resolution, you decrease the speed. These settings take effect the next time measurements are made.

NOTE

To see which settings are adjusted, you can set the Command setting of the Event Log to On. When command logging is on, each setting made by the Performance slider is listed as an Information event in the Event Log.

If the instrument is set to the DC voltage, DC current, digitize voltage, or digitize current function, changing the speed may change the function from the DC voltage or DC current function to the digitize voltage or digitize current function and vice versa.

When the temperature function is selected, the readings per second are shown as a range to accommodate the various transducer types. For additional information, see the Keithley Application Note *Increasing Production Throughput with Data Acquisition Systems*, document number 1KW-61369-0, which is available at tek.com/keithley (tek.com/products/keithley).

Can I close one channel and keep it closed while other channels are scanned?

If you close a channel, the channel remains closed while you scan if the closed channel is not in your scan. For example, if you close channel 103 and then scan from 101 to 105, channel 103 remains closed until your scan goes from 103 to 104, then it will open.

If you close channel 203 and run the scan, channel 203 remains closed. See the following examples.

Using the SCPI commands:

Send the commands:

```
ROUTE:SCAN:MODE USED
ROUTE:MULTiple:CLOSe (@203)
ROUTE:SCAN:CREate (@101:105)
```

Using the TSP commands:

Send the commands:

```
scan.mode = scan.MODE_OPEN_USED
channel.multiple.close("203")
scan.create ("101:105")
```

I see a command that is not in the manual. What is it?

You may see commands that are internal to the instrument if you:

- Have the event log set to record commands
- Capture commands with the save setup feature
- Store settings in a configuration list
- Set up a scan

If a command is not described in the Command Reference section, do not use it.

The commands that you might see include:

- `dmm.measure.configlist.set()`

Can the DAQ6510 have multiple user-defined RTDs in a scan list?

Yes, there can be multiple user-defined RTDs at one time.

What rack kit is required for mounting?

You can use the one of the following mount kits:

- Model 4299-10 Dual Fixed Rack-Mount Kit: Mount one DAQ6510 and one Series 26xxB instrument
- Model 4299-11 Dual Fixed Rack-Mount Kit: Mount one DAQ6510 and one Series 2400 or Series 2000 instrument
- Model 4299-8 Single Fixed Rack-Mount Kit
- Model 4299-9 Dual Fixed Rack-Mount Kit

Which rack kit is required to rack mount a DAQ6510 and 2280 side-by-side?

To rack mount a DAQ6510 and Series 2280 Precision Measurement DC Power Supplies side-by-side, you can use the Model 4299-10 Dual Fixed Rack-Mount Kit.

Next steps

In this section:

[Additional DAQ6510 information](#) 18-1

Additional DAQ6510 information

For additional information about the DAQ6510, go to tek.com/keithley (tek.com/products/keithley) for the most up-to-date information. From the website, you can access the following handbooks:

- *The Low Level Measurements Handbook: Precision DC Current, Voltage, and Resistance Measurements*
- *Semiconductor Device Test Applications Guide*
- *Switching Handbook: A Guide to Signal Switching in Automated Test Systems*

Access the DAQ6510 product page to find:

- *Data Acquisition Primer: An Introduction to Multi-Channel Measurement Systems*
- Application notes
- Updated drivers
- Updated firmware
- Software for use with the DAQ6510, including Keithley KickStart Software, which is coding-free instrument control software that includes a data acquisition and logging application

Your local Field Applications Engineer can help you with product selection, configuration, and usage. Check the website for contact information.

Specifications are subject to change without notice.
All Keithley trademarks and trade names are the property of Keithley Instruments.
All other trademarks and trade names are the property of their respective companies.

Keithley Instruments • 28775 Aurora Road • Cleveland, Ohio 44139 • 1-800-833-9200 • tek.com/keithley



Mess- und Prüftechnik. Die Experten.

**Ihr Ansprechpartner /
Your Partner:**

dataTec AG
E-Mail: info@datatec.eu
datatec.eu

04/2022